Sistema per a detectar o preveure violacions del SLA d'un servei en el núvol

Autor: Adrià Munuera Borrell

Director: Xavier Verdú Mulà
Co-director: Antonio Arellano Moral
Ponent: Alex Pajuelo González

Grau en Enginyeria Informàtica
Especialitat en Computació
2019

Resum

Aquest document conté el Treball de fi de Grau del Grau en Enginyeria Informàtica de l'alumne Adrià Munuera Borrell, especialitzat en Computació en la FIB (Facultat d'Informàtica de Barcelona).

En l'actualitat hi ha una tendència verificada de traslladar serveis al núvol, desacoblant-los de les capacitats de les màquines dels seus usuaris. Aquest canvi significa que un nou servei ha de ser capaç de satisfer la demanda provocada pels usuaris tot mantenint uns certs paràmetres de qualitat. Tot això està especificat i acordat entre el proveïdor del servei i client en un document anomenat SLA (*Service Level Agreement*). En aquest treball s'explora com mantenir sempre el compromís estipulat amb l'usuari en el SLA a través de detectar i preveure incompliments d'aquest gràcies a un entorn integral d'aprenentatge automàtic desenvolupat amb aquest propòsit.

Resumen

Este documento contiene el Trabajo de fin de Grado del Grado en Ingeniería Informática del alumno de Adrià Munuera Borrell, especializado en Computación en la FIB (Facultat d'Informàtica de Barcelona).

En la actualidad hay una tendencia verificada de trasladar servicios a la nube, desacoblandolos de las capacidades de las maquinas de sus usuarios. Este cambio significa que un nuevo servicio debe ser capaz de satisfacer la demanda provocada por los usuarios mientras mantiene unos estándares de calidad concretos. Todo esto está especificado y acordado entre el usuario y el proveedor del servicio en el documento llamado SLA (*System Level Agreement*). En este trabajo se explora como mantener siempre el compromiso estipulado con el usuario en el SLA a través de detectar y prever incumplimientos de este gracias a un entorno integral de aprendizaje automático desarrollado para este mismo propósito.

Abstract

This document contains the Master Thesis for the Bachelor's degree in Informatics Engineering of Adrià Munuera Borrell, a student with a Major in Computer Science by FIB (Facultat d'Informàtica de Barcelona).

Nowadays there is a verified trend to move services into the cloud, removing the dependency between the performance of a service and the computer capabilities of its end users. This change means that a certain new service must be able to satisfy the incoming demand generated by its clients while keeping a certain level of quality. This is specified and agreed between user and service provider in a document called SLA (*System Level Agreement*). In this document we explore how to always keep this agreement with the user using an integral machine learning environment which detects and predicts violations of the SLA. This environment has been specifically developed to fulfil this purpose.

Índex general

Índex de figures

Índex de taules

1. Introducció

1.1 Contextualització

En l'actualitat una de les tecnologies amb més previsió de creixement en el món de la informàtica és la basada en el núvol. Com que un servei basat en el *cloud* és un servei úbic que una persona pot fer servir en qualsevol moment, l'empresa proveïdora ha d'assegurar-se de que sempre pot mantenir els seus estàndards de qualitat. Un producte que s'executa en el núvol funciona sobre uns servidors que no tenen perquè estar pròxims físicament a l'usuari, i la qualitat del mateix es pot veure molt afectada per factors com per exemple la connexió a internet del dispositiu que es comunica amb el núvol, l'estat de la xarxa en tots els seus punts intermedis o l'estat dels servidors. El document anomenat **Acord de Nivell de Servei**[1] (*SLA, Service Level Agreement*) és el contracte on queda documentat quin és el nivell que l'empresa es compromet a oferir als seus clients. La **qualitat del servei**[2] (o *QoS, Quality of Service*) és la mesura de si aquest Acord de Nivell de Servei s'està complint de manera satisfactòria o no.

El monitoreig de dades en una empresa pot ser enormement beneficiós per a aquesta. A part de que molts negocis generen dades aprofitables, hi ha optimitzacions possibles en base a aquesta nova informació. Per exemple, una empresa que es dediqui a les reparacions, si es capaç de monitoritzar correctament els seus treballadors disponibles i les seves feines a dur a terme, pot planificar millor els encàrrecs en relació a l'espai i al temps i estalviar-se diners fent feina de manera més eficient. En un servei en el núvol aquestes dades podrien fer-se servir, per exemple, per a quan un ordinador estigui a punt d'arribar a la seva màxima capacitat donar-ne part de la seva càrrega de feina a un segon; o, donada una hora concreta i una demanda estimada d'usuaris, preparar-se per a la pujada de feina del servei engegant nous servidors o estalviar diners apagant màquines en la baixada de la demanda.

Totes aquestes dades es generen en grans quantitats, donat a que avui en dia hi ha molts dispositius connectats a la xarxa que són capaços d'obtenir un gran ventall de dades diferents i enviar-los a un servidor. La mida d'aquesta informació pot ser tant gran que les tècniques més ortodoxes de processament es queden antiquades, i s'han creat sistemes especialitzats per a processar el gran flux de dades. A aquest concepte se l'anomena ***Big Data***.

En concret, en els serveis en el núvol, hi ha moltes dades i molt variades que són generades per un servei *cloud*, pel que és costós detectar d'una forma manual o tradicional quan hi ha errors i perquè. A més, molts serveis recullen dades amb una certa latència afegida, és a dir que no són en temps real. Això és un factor afegit que dificulta l'automatització del procés perquè no sempre es pot tenir al moment les dades que un necessita per a treballar.

Aquestes dades, un cop en un servidor de destí, es poden processar per a usar-les de diferents maneres. Per exemple, existeixen algorismes que detecten patrons en la informació

o que són capaços de fer estimacions amb una precisió estadísticament segura. Un conjunt important d'aquestes tècniques són les vinculades amb l'**aprenentatge automàtic** (*Machine Learning* o *ML* en anglès).

Dins d'aquest camp d'estudi hi ha dos grans tasques que realitzen els algorismes: classificació de dades, on el que s'intenta és decidir a quina d'un conjunt de categories pertanyen unes dades concretes; i regressió de dades, on en base a unes dades inicials intentem predir resultats del futur. Aquests algorismes però poden necessitar ser entrenats primer, és a dir, requereixen d'informació per a trobar una sèrie de paràmetres matemàtics que minimitzin l'error de l'aproximació que ells mateixos suposen amb les dades proporcionades.

Per acabar, dir que personalment he pogut treballar en una empresa de joc en el núvol (Ludium Lab) gràcies a un conveni de col·laboració amb la universitat i he pogut constatar el context i els problemes que explico en aquest document.

1.2 Formulació del problema

Alguns dels serveis o productes que fan ús de la tecnologia en el núvol de forma integral poden necessitar pactar amb el client quina és la qualitat del servei que proveirà (SLA), ja que hi ha una gran diversitat de factors que la poden afectar i estàn fora de la possibilitat de que el proveïdor en doni solució per la pròpia naturalesa ubiqua del núvol.

Degut a les característiques dinàmiques d'ús dels serveis ubics del núvol es genera un gran volum d'informació que en el seu conjunt pot presentar detalls de les violacions del SLA. Precisament, per la naturalesa d'aquestes dades no és viable processar-les manualment, i per tant convé automatitzar el procés de detecció de violacions en el SLA per a poder avançar-se als problemes i actuar-ne en conseqüència, buscant-ne la seva prevenció i detecció. Actualment no hi ha eines en el mercat que es puguin adaptar de manera directa a les necessitats diverses d'aquests serveis.

1.3 Objectius del projecte

L'objectiu principal d'aquest treball és analitzar, dissenyar, implementar i validar un sistema d'aprenentatge automàtic configurable i extensible que detecti o prevegi violacions del SLA d'un servei en el núvol.

En aquest treball es pretén poder donar resposta a aquesta problemàtica creant un entorn fàcilment utilitzable per un servei en el núvol. Aquest programari hauria de ser fàcilment extensible a diversos entorns que proporcionin dades provinents d'un servei en el núvol. A més, volem que el sistema estigui basat en tècniques d'intel·ligència artificial, i en concret, en algorismes d'aprenentatge automàtic, per a poder automatitzar el procés de detecció de violacions del SLA.

Per a tal d'arribar a aquesta meta hi ha altres objectius secundaris que s'han de prendre en consideració:

1. Adaptar les dades que obtenim en cru, processant-les de forma adequada per a l'entrenament dels models d'aprenentatge automàtic i l'extracció d'informació.
2. Fer ús de llibreries d'algorismes de *machine learning* amb diferents mètodes per a tenir gran varietat de models i paràmetres per elegir.
3. Preparar l'entorn, entrenar i comparar quins són els millors algorismes i amb quins paràmetres per a obtenir els millors resultats possibles.
4. Per a verificar el bon funcionament del sistema es pretén diferenciar dos casos complementaris en que es pot detectar o preveure incompliments del SLA:
   - **Cas A:** detectar quan en un servei en el núvol cal modificar-ne la mida del conjunt d'ordinadors que l'integren en funció del nombre d'usuaris.
   - **Cas B:** detectar quina és la proporció d'ús dels diferents serveis o programes que ofereixi un *cloud* en un moment determinat per a aplicar optimitzacions juntament amb els models creats en el cas anterior.

Els casos A i B són casos representatius d'escenaris reals per a alguns serveis en el núvol, com és el cas de Ludium Lab. En aquest document utilitzarem dades de la plataforma de videojocs Steam per a poder validar-ne els resultats, tal com es detalla a la Secció **4.2** d'aquest document. És important destacar que la empresa Ludium Lab té la intenció de valorar l'ús de l'entorn desenvolupat en aquest treball.

1.4 Actors implicats

1.4.1 Desenvolupador

L'encarregat d'analitzar, dissenyar, implementar i comprovar la qualitat del sistema seré jo mateix. En l'empresa Ludium Lab vaig ocupar primer el càrrec de desenvolupador encarregat d'Android i HTML5, i ara sóc l'encarregat de Ciència de les Dades.

1.4.2 Tutors del projecte

Els director d'aquest TFG és el Xavier Verdú Mulà, professor del departament d'Arquitectura de Computadors de la Facultat d'Informàtica de Barcelona. El co-director és l'Antonio Arellano, treballador de Ludium Lab, encarregat de desenvolupar el rol de *Site Reliability Engineer* (SRE) en l'empresa. El ponent del projecte és l'Alex Pajuelo Gonzalez, també professor del departament d'Arquitectura de Computadors. Amb ells tres hi haurà un seguiment setmanal del projecte per a la seva correcta execució.

1.4.3 Proveïdor del servei en el núvol

És el potencial usuari d'aquesta eina. Si l'adopta, el servei es podrà beneficiar del sistema proposat en aquest treball i podrà adoptar mesures per a millorar el compliment del SLA.

2. Estat de l'art

Actualment, i donat a l'augment de serveis en el núvol, s'està investigant força sobre com combinar el camp de l'aprenentatge automàtic amb la detecció de violacions del SLA de forma pública. En el document d'en Sahar Mohamed Musa et al. [4] es proposa un algorismes per a detectar violacions del SLA. En altres articles com en el de Tong-Sheng Wong et al. [5] s'explica una solució basada en *Machine Learning* i regles de decisió que aporta molt bon rendiment, arribant a crear un sistema basat en un Màquina de Vectors Suport (*Support Vector Machine, SVM*) amb una precisió d'un 99% d'encert segons els seus resultats. Aquest últim article conté idees extrapolables a aquest TFG, com l'ús de regles de decisió junt amb un model d'aprenentatge automàtic per a detectar les violacions del SLA. Cal destacar que de forma pública no s'han trobat referències a sistemes funcionals dins d'empreses que compleixin els objectius marcats en aquest treball. El que sí es sap, és que les solucions actuals a aquest problema[6] són poc flexibles i poc portables, entre altres raons perquè cadascú es fa el programari segons les seves necessitats.

Per a sistemes en el núvol no només s'estudien les violacions del SLA, sinó que hi ha moltes mètriques crucials per a assegurar el bon funcionament d'aquests sistemes i augmentar-ne la seva tolerància a errors. Un exemple de com s'aplica l'aprenentatge automàtic a diferents tipus d'errors és l'article [7], on a part de fer servir SVMs també fan ús de tècniques de regressió lineal per a preveure errors.

També cal esmentar en aquest apartat que ja hi ha diferents serveis que ofereixen *Machine Learning as a Service*, és a dir, que ofereixen poder treballar amb aquesta tecnologia de la intel·ligència artificial en el núvol a través d'una aplicació. Exemples d'aquest producte són Google Cloud AI o Amazon Web Services Machine Learning, de fàcil ús i desplegament. El que sí que és cert és que aquests serveis estan en núvols de tercers, pel que s'han de pujar-hi les pròpies dades per a fer-ne ús. A més, estan orientats a *Deep Learning*, que en justifica la seva potència. En el cas que ens ocupa però, i tot i la seva conveniència, aquestes plataformes no permeten ni la execució en local, ni el total control sobre dades del nostre servei ni una modularitat total que un proveïdor de serveis en el núvol pot aprofitar segons les seves necessitats.

Per alta banda, si que existeix programari per a monitorar el compliment del SLA[8], però no en fan prediccions del seu compliment per a avançar-se als problemes que es puguin presentar.

En aquest apartat cal esmentar també el TFG de l'Antonio Arellano [9], graduat de la FIB, co-director d'aquest TFG. En el document que ell va elaborar es proposa una solució per a recollir dades crucials en el servei de *Cloud Gaming* de Ludium Lab. En la mesura de que el present TFG necessita dades per a poder entrenar els algorismes d'aprenentatge automàtic el treball realitzat per l'Antonio és una sòlida base com a model d'un sistema per a obtenir la informació desitjada.

3. Definició de l'abast

3.1 Meta a assolir

El *software* desenvolupat en aquest projecte ha de ser un entorn fàcilment extensible i configurable i suficientment generalista per a que un *cloud service* qualsevol pugui integrar-s'hi seguint unes pautes donades. El programari creat ha de complir amb els requisits proposats en la secció **1.3** d'objectius d'aquest document. Idealment, aquest programa ha de poder comunicar-se amb altres programes fàcilment per a que les prediccions fetes per aquest es puguin integrar en altres algorismes.

El programari ha d'avisar quan detecti o prevegi violacions del SLA. La presa de decisions en funció dels resultats obtinguts gràcies als algorismes d'aprenentatge automàtic queden fora de l'abast d'aquest TFG i en mans de l'usuari final.

3.2 Possibles obstacles

En el desenvolupament d'aquest treball hi ha diversos obstacles que s'han de treballar i vigilar. Els principals són els següents:

- Definir i obtenir quines dades s'utilitzaran per als casos de validació.
- Realitzar un processament correcte de les dades per a poder extreure les característiques adequades per a alimentar els algorismes de *machine learning*.
- Trobar i reparar errors en el codi (*bugs*).
- Modelar el temps com una característica intrínseca de les dades, on l'ordre en que es rep la informació importa. Això significa que s'haurà de fer servir models d'algorismes que tinguin en compte el temps, o processar les dades d'alguna manera per expressar la característica temporal.
- Trobar quina solució d'aprenentatge automàtic és la que té un rendiment més elevat en una varietat de situacions suficientment representativa, tant per al Cas A com per al B.
- El fet de poder trobar nous casos on detectem violacions del SLA però no ho prevenim dona peu a tenir noves dades per a re-entrenar els algorismes d'aprenentatge automàtic. El procés d'entrenar un model de ML és la part més costosa en temps de processament relacionada amb aquests algorismes, pel que s'haurà de tenir en compte a l'hora de tornar a entrenar amb les noves dades, ja que haurem de buscar models ràpids d'entrenar.

3.3 Valoració d'alternatives i pla d'acció

Segons com s'han especificat els possibles problemes del projecte en la secció anterior, a continuació es detalla la solució dels principals obstacles que es poden donar en el transcurs del desenvolupament del projecte:

- En el cas del processament de dades per a extreure característiques correctament, la tasca de visualització de les dades i la d'entrenament i validació del rendiment del

sistema ens permetrien orientar en com s'han de processar les dades i fer petits retocs a l'algorisme si escau, respectivament.

- En el cas dels errors de codi ja s'ha contemplat en la planificació temporal l'existència d'aquests, ja que sempre que es desenvolupa programari hi ha alt risc de crear-ne algun.
- Per a poder modelar el temps com a característica intrínseca de la informació recopilada s'estudia en la fase d'aprenentatge tant com processar les dades tenint-lo en compte com els diferents algorismes que el contemplen. Això hauria de permetre treballar amb aquesta característica de forma correcta.
- Per a evitar que cadascun dels algorismes d'aprenentatge automàtic que decidim usar com a models finals per a cada cas de validació triguin molt a entrenar-se, es tindrà en compte aquest temps per a intentar minimitzar-lo, tot equilibrant la compensació entre el rendiment d'un model i el temps d'entrenament d'aquest.

4. Metodologia seguida

La metodologia a seguir en aquest projecte serà l'anomenada cascada retroalimentada (Figura 1) [10].

Aquesta metodologia consisteix en les fases estipulades en la figura inferior. És una bona metodologia pels propòsits d'aquest TFG perquè el projecte està definit en unes fases clares i té uns objectius definits (i per tant un rendiment mesurable). A més, no hi ha tasques que es puguin fer en paral·lel ni que no tinguin dependències de les anteriors a elles: per exemple, no podem començar a validar el sistema fins a tenir-lo completament acabat, doncs no podem construir un prototip del projecte.

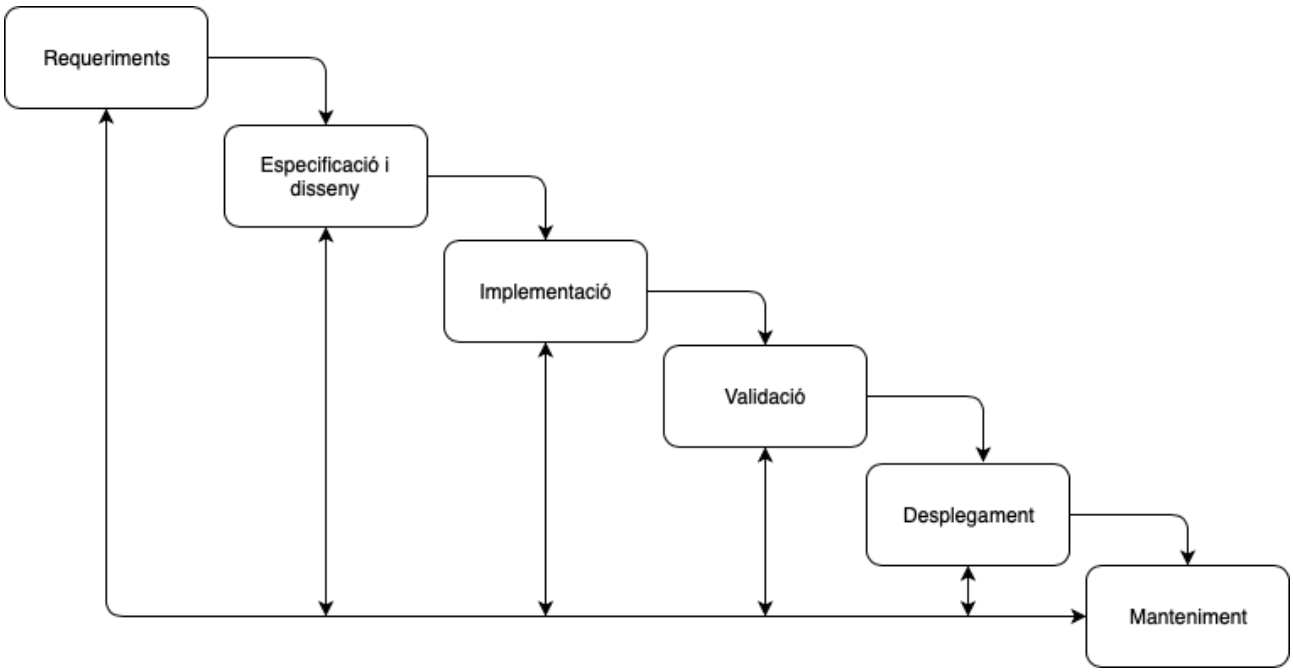

Figura 1. Metodologia cascada retroalimentada

Sí que podem iterar sobre diferents versions amb les mateixes característiques, i per això s'ha decidit que és important que el model segueixi la cascada però retroalimentada. Per exemple, pot ser necessari modificar el processament de dades per a adequar-lo més a la fase d'entrenament.

Aquí cal deixar clar que a diferència de les metodologies àgils, que es centren en crear un programari concret expandint les seves característiques cada iteració, el *software* que es realitzarà a arrel d'aquest TFG no es pot construir diferenciant característiques i afegint-ne cada cop més (a excepció d'afegir nous models d'aprenentatge automàtic) doncs la implementació de l'entorn requereix la creació d'un conjunt de classes grans que si no estan programades un cop es comenci el procés de validació, no té sentit dur-lo a terme. En el cas present, les iteracions seran per a resoldre errors del codi o millorar les dades.

4.1 Mètodes de seguiment

Per a poder fer un correcte seguiment del progrés en el treball s'utilitzaran principalment les tres eines següents:

- Per a portar el control de l'estat general del projecte i les seves tasques, s'utilitzarà el *software* Trello (trello.com). Aquest programa és una eina gratuïta que permet visualitzar les tasques com a tiquets classificades segons la seva prioritat i si estan sent resoltes o no.
- Per a controlar les versions del *software* que es desenvolupi, es farà ús de git. El portal github.com disposa de repositoris privats en el núvol sense cap cost per a estudiants.
- Per a consultar i mantindre les referències fetes servides en aquest projecte, s'utilitzarà el programari Mendeley.

4.2 Metodologia de validació

Per tal de validar el sistema creat, amb aquest i les dades de Steam es pot simular els dos casos explicats anteriorment en l'apartat **1.3** d'objectius. Les dades per a aquest procés es poden obtenir al web steamdb.info, però només per a la última setmana ocorreguda. Per tant, per a la realització d'aquest treball s'han anat descarregant dades de forma setmanal, i estan disponibles a https://www.dropbox.com/s/1asqdnpc9xnymkz/Data.zip?dl=0.

La validació dels casos s'ha dut a terme comparant l'exactitud dels diferents mètodes d'aprenentatge automàtic seleccionats per tal d'obtenir el millor model per a ambdós i comprovant que es podia arribar a complir els objectius d'aquests, tot assegurant-se que l'entorn de ML desenvolupat no contenia errors. Per a més detalls, consultar les seccions de validació de l'entorn desenvolupat i dels dos casos.

5. Desenvolupament de l'entorn de creació, entrenament, testeig i predicció amb aprenentatge automàtic *Cronos*.

Per entendre la raó de desenvolupar aquest sistema, considerem el cas d'una organització que genera dades i que vol fer ús d'aquestes alimentant-ne algorismes de ML per a poder fer prediccions útils per a ells i els serveis que ofereixen. Idealment, i més en el cas d'una empresa de tecnologia, es valora la facilitat d'ús del sistema, la privacitat de les dades que es fan servir per alimentar-lo; i en general, el control absolut sobre tot el que hi passa. És per això, que, tal com es va veure en el Capítol **2** Estat de l'Art d'aquest document, en aquest treball ens hem plantejat el repte de crear un sistema integral de ML que permeti fer ús de dades i algorismes de la forma més modular i personalitzable possible, sense deixar de banda la senzillesa. Aquesta és la raó de ser del sistema desenvolupat, anomenat *Cronos*[1].

5.1 Anàlisi

Com a primer pas de la fase de desenvolupament és essencial que analitzem quines són les característiques que s'espera que aquest sistema compleixi en funció de les necessitats dels seus usuaris. A continuació es raonaràn i s'exposaran diferents funcionalitats i conseqüències de les mateixes que assentaran la base per a després poder enumerar tot el que s'esperi que *Cronos* faci.

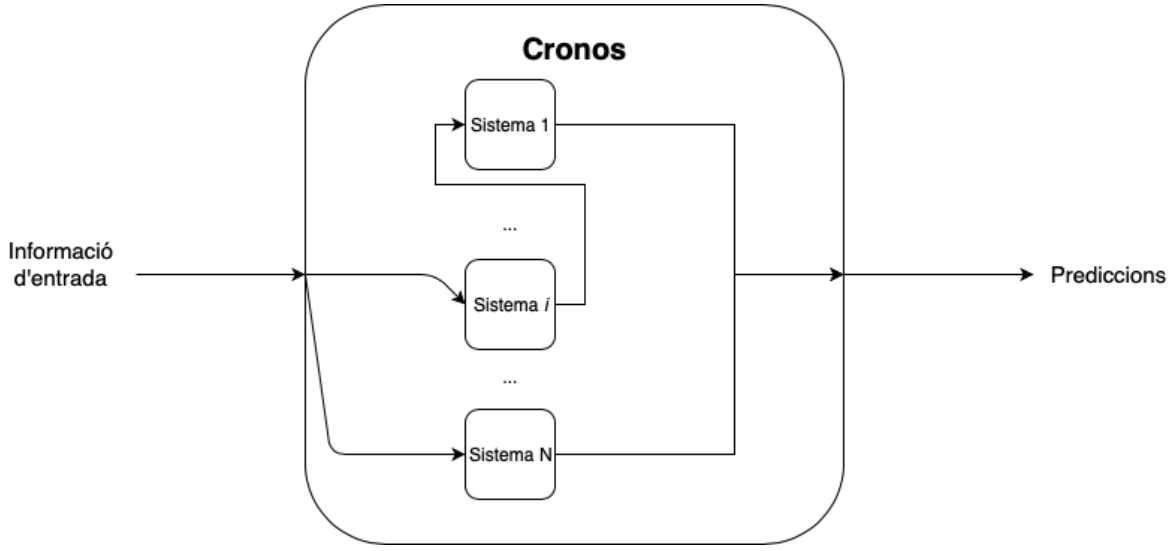

Figura 2. Esquema general del sistema *Cronos*

[1] De la mitologia grega, titià senyor del temps i pare dels deus de l'Olimp.

El sistema que volem crear és un que permeti configurar de forma fàcil diferents models d'aprenentatge automàtic. L'objectiu de cada algorisme entrenat és poder fer prediccions i servir-les fàcilment a altres algorismes o sistemes que les puguin necessitar (Figura 2).
Però, prèviament a poder fer cap predicció cada sistema ha de ser entrenat segons uns paràmetres a concretar (els híper-paràmetres del model) de tal forma que hem de ser capaços d'especificar una gran varietat de configuracions diferents per a cada model, i trobar-ne quina és la millor per a les nostres dades. Per a valorar com de bo és cada sistema també és necessari que implementem una forma fàcil de procedir a aplicar-hi tests. En l'entorn *Cronos* hauríem de poder saber com de bé un sistema prediu les dades de testeig amb les d'entrenament i quin és l'error associat a aquesta validació.

Cada model que entrenem ha de poder ser guardat, ja que cada cop que vulguem usar-ne un no volem haver d'entrenar-lo un altre cop perquè és car computacionalment parlant. També hauríem de poder exportar els models entrenats d'una forma senzilla, per a no haver de re-entrenar un algorisme amb el mateix propòsit dues o més vegades en diferents ordinadors. També, en el cas de les prediccions, hem de ser capaços d'entregar dades a partir de les quals predir, i després extreure'n el resultat. Aquesta informació base per a predir ha de ser fàcilment subministrable, i el resultat ha de poder ser recollit posteriorment per qualsevol altre sistema que les necessiti. Per tant, l'usuari ha de tenir el control total tant de la font de la informació d'entrada al sistema com de les dades de sortida.

A més, les dades han de ser modelades amb funcions de pre-processament i processament. Com que podem especificar una infinitat d'aquestes funcions, no hi ha cap forma predeterminada de configurar-les. És més, les mateixes dades un cop modelades poden alimentar algorismes amb propòsits molt diferents; tal com demostrarem amb els dos casos estudiats en aquest TFG, que es nodreixen de les mateixes dades i prediuen informació diferent (encara que relacionada). Per tant, aquestes dues funcions per a transformar les dades s'han de poder afegir a l'entorn principal com a petits programes fàcilment importables, i han de ser usables si segueixen un estàndard marcat.

L'entorn, encara que no estigui executant-se, ha de poder fer prediccions amb les dades d'entrada i servir-les. Per tant, cal trobar alguna forma d'automatitzar la crida a la funció de predicció per a tots els sistemes que estiguin entrenats i que vulguem obtenir-ne prediccions.

De totes les idees i demandes exposades, s'extreu el següent conjunt de característiques per a l'entorn *Cronos*:
- Configuració fàcil de diferents models de *Machine Learning* amb diferents paràmetres i algorismes.
- Poder especificar quines dades són les d'entrenament i quines són les de validació.
- Capacitat per a validar els diferents models amb dades de test i extreure'n l'error associat.

- Persistència dels models configurats i entrenats. Capacitat per a exportar-los a altres ordinadors que executin *Cronos*.
- Capacitat d'especificar quines són les dades a partir de les quals predir i recollir el resultat que aquestes han generat de forma que altres entitats puguin reutilitzar-los (fins i tot altres sistemes). Donar el control d'aquesta informació a l'usuari.
- Poder especificar qualsevol funció de pre-processament i processament de les dades per a cada model, responent a la infinitud de possibilitats que tenen aquestes.
- Poder fer prediccions amb els models habilitats i entrenats amb aquesta fi, encara que l'entorn no s'estigui executant en aquell precís instant.

Per acabar, i donat a que no es necessita tenir una interfície gràfica per a gestionar constantment tots els models, l'aplicació desenvolupada serà per a línia de comandes. També el fet de desenvolupar una interfície gràfica que sigui correcte comportaria invertir-hi molt de temps que és millor utilitzar per a programar i comprovar funcionalitats.

5.2 Especificació

5.2.1 Especificació de les funcionalitats

Donades totes les característiques ja definides en la secció immediatament anterior, cal trobar com donar-hi resposta. Per tant, en aquesta secció s'explicarà de forma detallada que es vol fer de tot el que ja s'ha definit que s'ha de fer de forma ampliada i més concreta, sense entrar en detalls específics d'implementació:

5.2.1.1 Configuració fàcil dels diferents models. Especificar dades d'entrenament, validació i entrada. Especificar lloc on deixar les prediccions

Hem de poder crear nous models i configurar-los de forma senzilla. Configurar-los implica poder especificar quines són les dades d'entrenament, les de validació i les d'entrada; i també a on es deixen les dades predites a partir de les últimes. Les dades predites han de poder ser utilitzades com a entrada també per altres models dins de *Cronos*. A més, s'ha de poder configurar quines són les funcions de pre-processament i processament. Cada sistema s'ha de poder entrenar d'acord amb tots els diferents paràmetres per a configurar l'algorisme d'aprenentatge automàtic escollit, de tal forma que múltiples combinacions d'híper- paràmetres siguin possibles.

5.2.1.2 Validació de models

Un cop entrenat, cada model ha de poder ser validat per a saber com de bona ha sigut la solució aconseguida gràcies a les funcions de pre-processament i processament, l'algorisme elegit i els paràmetres del model. S'ha de poder obtenir quin és l'error associat a la predicció de les dades de validació amb l'algorisme ja entrenat; i si fa falta, hem de poder re-entrenar el model per a intentar obtenir una rebaixa de l'error. D'aquesta forma podem obtenir el millor model possible.

5.2.1.3 Persistència dels models

Com que el procés de configuració i entrenament, encara que senzill, consumeix temps físic i de computació, cal implementar un sistema que mantingui els valors ja elegits per a cada sistema i el model ja entrenat. Idealment, això ha de ser aconseguit amb un fitxer de dades:

d'aquesta forma en maximitzem la seva portabilitat a altres entorns *Cronos*, per a no haver de re-configurar i re-implementar tots els models que vulguem exportar. Per tant, a cada nova execució de *Cronos* hem de carregar tots els sistemes ja construïts; i anar guardant els progressos sobre els existents a mesura que hi anem treballant.

6.3.1.4 Especificar funcions de pre-processament i processament

Les funcions per a mantenir la integritat de les dades pel model (pre-processament) i modificar-les per a que aquest les entengui (processament) són molt variades i poden ser molt complexes, pel que no és possible crear un sistema per a poder especificar quines de totes les possibles volem. Per tant, es crearà una forma per a poder acoblar aquestes dues funcions de forma transparent a l'entorn, encara que aquestes hauran de ser programades a part (i depenent de la implementació, compilades). Es proporcionarà una especificació de quins són els paràmetres proporcionats a les funcions i quins valors han de retornar per a que funcioni tot correctament.

5.2.1.5 Fer prediccions de forma automàtica

Per a que l'entorn sigui completament útil i usable en producció és necessari que encara que aquest no estigui en la llista de processos obert s'executi de forma automàtica la generació de noves prediccions a partir de les dades d'entrada ja especificades en la configuració de cada sistema. Això ha de fer-se en cada sistema que nosaltres ja tinguem correctament entrenat i configurat. A més, hem de tenir la capacitat d'habilitar i deshabilitar per a cada model si aquest pertany al conjunt sobre els que volem executar prediccions automàtiques o no.

5.2.2 Classes del programa

Per altra banda, pels requisits explicats, ja podem donar un primer model de quines són les classes que integraran l'entorn *Cronos*. Aquest sistema s'il·lustra en el següent gràfic:

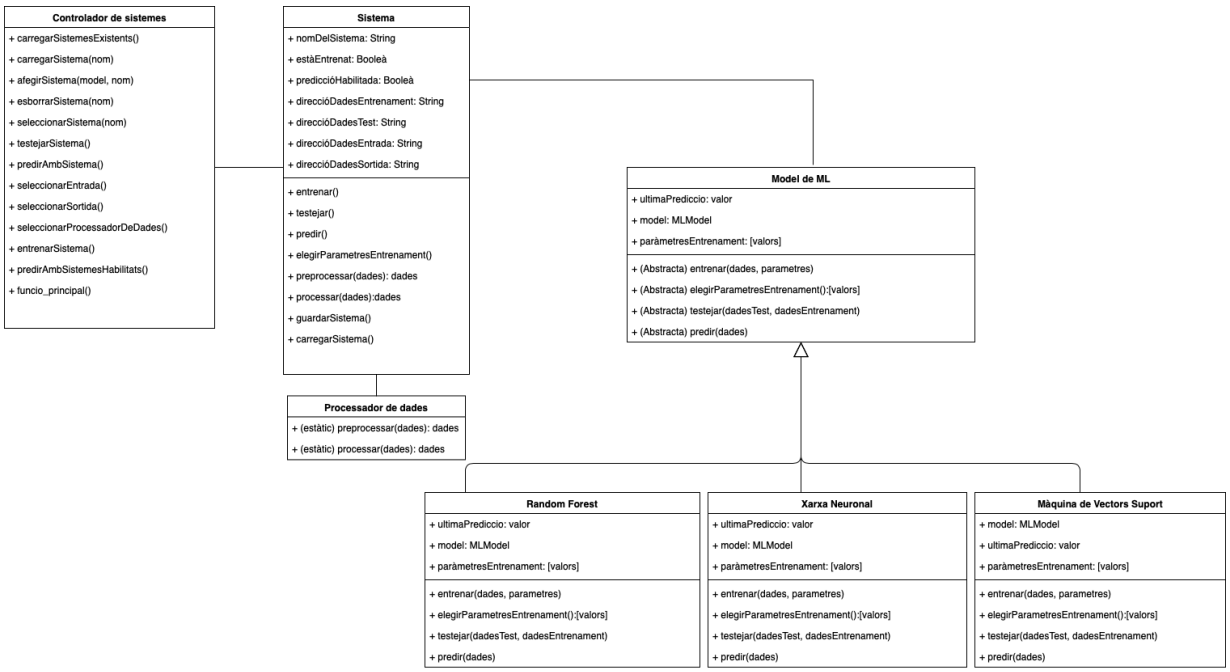

Figura 3. Gràfic amb les diferents classes de l'entorn *Cronos* i la seva relació entre elles

El propòsit general de cada classe es es defineix a continuació:

- **Controlador de sistemes.** Controlador de la resta de classes, fa possible l'ús de l'entorn i la interacció amb els diferents sistemes. Permet, per tant:
  - Crear nous sistemes.
  - Eliminar sistemes.
  - Carregar els sistemes existents.
  - Cridar a les diferents funcions de cada sistema.
  - Llistar tots els sistemes.
- **Sistema.** Classe que conté diferents atributs per a assegurar el bon funcionament d'un sistema (camins fins als diferents arxius que es necessiten per a un bon funcionament, valors indicant l'estat del sistema), a més d'un processador de dades i un model de ML. Permet interaccionar amb el model i cridar a les funcions del processador de dades. També s'encarrega de guardar-se i carregar-se a si mateixa.
- **Processador de dades.** Classe que implementa les funcions de pre-processar i processar dades. Aquesta classe l'ha de proporcionar l'usuari per a cada sistema.
- **Model de ML.** Classe abstracta que defineix quins atributs i mètodes ha de tenir una subclasse d'aquesta per tal de que funcioni correctament junt amb la classe Sistema.
- **Subclasses de Model de ML.** Implementen totes les funcionalitats especificades en la seva classe pare.

5.3 Implementació de *Cronos*

5.3.1 Llenguatge de programació triat

Per a implementar *Cronos* s'ha triat el llenguatge de programació Python[11]. Aquest llenguatge interpretat ofereix un gran ventall d'avantatges per als programadors, i actualment és un dels més usats i més valorats per aquest col·lectiu.

Python ofereix una sintaxis clara i molt llegible que en facilita tant la creació de codi com la busca i eliminació d'errors. A més, i per a fer encara més fàcil el procés d'implementació dels programes, Python disposa d'una llibreria estàndard gran i la possibilitat d'instal·lar ràpidament mòduls nous creats per la comunitat gràcies a la eina `pip`[12]. Això fa que abans d'escriure una nova funcionalitat complexa sigui convenient (i molt senzill) integrar la solució escrita per una altra persona. Més endavant en aquesta secció d'implementació parlarè de quines són les llibreries usades en el meu projecte.

També cal destacar que aquest llenguatge, al ser interpretat, és multiplataforma: per a funcionar en un ordinador executant un cert sistema només necessitem l'intèrpret i el codi a executar. Existeixen intèrprets de Python per als principals sistemes operatius: Windows, MacOS, Linux i totes les seves distribucions.

Per últim cal dir que Python té un sistema de tipus dinàmic que implica que no hem de definir quin tipus de variables és cadascuna perquè automàticament es dedueix. Això fa que una variable en diferents moments de l'execució pugui pertànyer a classes diferents. El

sistema de tipus de Python facilita el procés de desenvolupament; però recau en el programador la responsabilitat de que un programa s'executi correctament: cal vigilar que en cap moment s'intenti fer un ús d'una variable com si fós d'una classe que no és la seva[2].

5.3.2 Classe Controlador de Sistemes

Aquesta és la classe que s'executa al inicialitzar l'entorn, doncs és la que conté la funció *main()*. La implementació d'aquesta classe no és complicada perquè és un intermediari entre les comandes de l'usuari i els sistemes d'aprenentatge automàtic creats. A partir d'una interfície en la consola es permet crear, destruir i manipular sistemes.

La llista de sistemes creats es guarda en un fitxer, on s'especifica el nom de cada sistema. Aquesta llista és la que després ens permetrà carregar sistemes anteriorment guardats.

5.3.3 Classe Sistema

La classe Sistema integra el model d'aprenentatge automàtic amb tots els paràmetres i funcions per a controlar-lo. La implementació de l'algorisme de ML està en les subclasses de la classe "Model de ML", però elements com si està entrenat o quin algorisme de predicció es fa servir estan en aquesta classe. Per tant, *Sistema* conté tota la informació necessària per executar correctament un model d'aprenentatge automàtic però sense entrar a programar la part específica de ML.

5.3.3.1 Persistència dels sistemes creats

Un dels requeriments detectats en aquest projecte és la necessitat de guardar i carregar els diferents sistemes que tinguem en l'entorn *Cronos*, per no haver-los ni de crear ni d'entrenar repetidament. A més, si els sistemes ja entrenats són exportables a altres ordinadors executant *Cronos* encara es guanya més valor.

Per tant, per a abordar aquesta demanda s'ha utilitzat la llibreria de Python *Pickle*[13]. Aquesta llibreria, instal·lable gràcies a `pip`, permet guardar i carregar instàncies de classes en fitxers. El que s'ha programat és que en qualsevol moment que hi ha un canvi en el Sistema de nom *NomSistema*, es guarda l'estat en un fitxer anomenat NomSistema.pickle. Després, gràcies a la mateixa llibreria, es pot carregar l'estat de la classe buscant el fitxer NomSistema.pickle. Aquesta és la raó per la qual la classe *Controlador de Sistemes* ha de mantenir una llista dels sistemes existents, perquè a l'hora de carregar un sistema es busca l'arxiu .pickle del seu mateix nom.

5.3.3.2 Especificació, lectura i escriptura de les dades del model

La classe *Sistema* també té paràmetres per a indicar quin és el camí fins als fitxer d'entrada, sortida, entrenament i validació de cada model d'aprenentatge automàtic. També té les funcions per a interactuar amb els mateixos, el que significa que implementa les funcions de lectura i escriptura d'aquests fitxers. És important remarcar que la entrada i sortida es fa a través de fitxers per a cedir el control de les dades de forma total a

[2] Existeixen formes de comprovar quin és el tipus d'una variable i actuar-ne en conseqüència.

l'usuari, on fins i tot hi poden haver programes externs que escriguin i llegeixin en les dades del sistema *Cronos*, alimentant-lo o fent ús de les seves prediccions.

Aquí cal remarcar que una decisió d'implementació ha sigut el format d'aquestes dades: cada fila de dades ha de ser un conjunt de caràcters on cada valor diferent va separat per un punt i coma, i no poden existir ni cometes simples ni dobles en aquests fitxers. Això és així perquè les funcions de lectura l'únic que fan és llegir cada fila i transformar cada valor en una cadena de caràcters, perquè el sistema a priori no sap quin és el tipus de cada dada. En canvi, és l'usuari qui sí sap que significa cada dada individual, i com que ell serà l'encarregat de programar les funcions de pre-processat i processat de dades, ell sí hi aplicarà el seu coneixement sobre els tipus de dades. Un exemple de dades correctament formatades per a que *Cronos* les utilitzi seria el següent:

```
2019-04-28 10:30:00;26656;26974,52777777777;2975
2019-04-28 10:40:00;26974;26905,077304261642;2842
2019-04-28 10:50:00;27295;26835,111166171115;2830
2019-04-28 11:00:00;27572;26765,23035733949;2798
2019-04-28 11:10:00;27925;26695,047072183672;2812
```

En aquest cas concret, la funció de lectura crearia una llista de llistes on cada llista dins de la llista de llistes conté diferents cadenes de caràcters. Després l'usuari en les funcions de transformació de les dades especificarà, per exemple, que la primera dada de cada fila és una data.

5.3.3.3 Interacció amb el model

Per últim, la classe Sistema també permet interaccionar amb el model d'aprenentatge automàtic triat. Més enllà de cridar les funcions d'entrenament, predicció, testeig i tria de paràmetres sobre aquest, també permet la modificació d'altres paràmetres:

- Si el model ha estat entrenat, permet activar o desactivar la capacitat de fer prediccions de forma automàtica sobre aquest model.
- També té la capacitat de canviar l'algorisme que fa servir el sistema, creant un nou objecte de Model de ML que associar-se a si mateix. Aquesta opció, evidentment, reinicia els indicadors de si un model està entrenat i de si es vol fer prediccions automàtiques sobre ell o no.

5.3.4 Models de ML implementats

Per a aquesta implementació de *Cronos*, s'ha decidit implementar tres mètodes d'aprenentatge supervisat de regressió: xarxes neuronals, *Random Forests* i màquines de vectors suport. S'han elegit aquests tres algorismes perquè els dos casos que es volen estudiar com a validació de l'entorn són de regressió; però tal com està especificat el sistema caldria fer canvis mínims per a afegir models nous.

Per a fer ús d'aquests mètodes en el programa es fa servir la llibreria *scikit-learn*[14], instal·lable gràcies a `pip`. Aquesta llibreria conté moltes implementacions de diferents algorismes d'aprenentatge automàtic, i entre aquestes, les dels models desitjats. Això ha permès la implementació de les tres subclasses de la classe *Model de ML* explicada en la secció d'especificació del sistema.

Per tant, s'ha pogut implementar tant l'elecció de paràmetres del model, com l'entrenament, la predicció o el testeig d'un sistema per a cadascun dels tres models. En el cas de la funció de validació, aquesta mostra l'error de les dades de *test* amb el model entrenat amb els paràmetres elegits i les dades de *train*, a més de mostrar la diferència entre les dades de validació i els valors predits en un gràfic. Si per exemple, entrenéssim amb unes dades donades una màquina de vectors suport i volguéssim saber com de bo és el model creat, es generaria una gràfica similar a la següent:

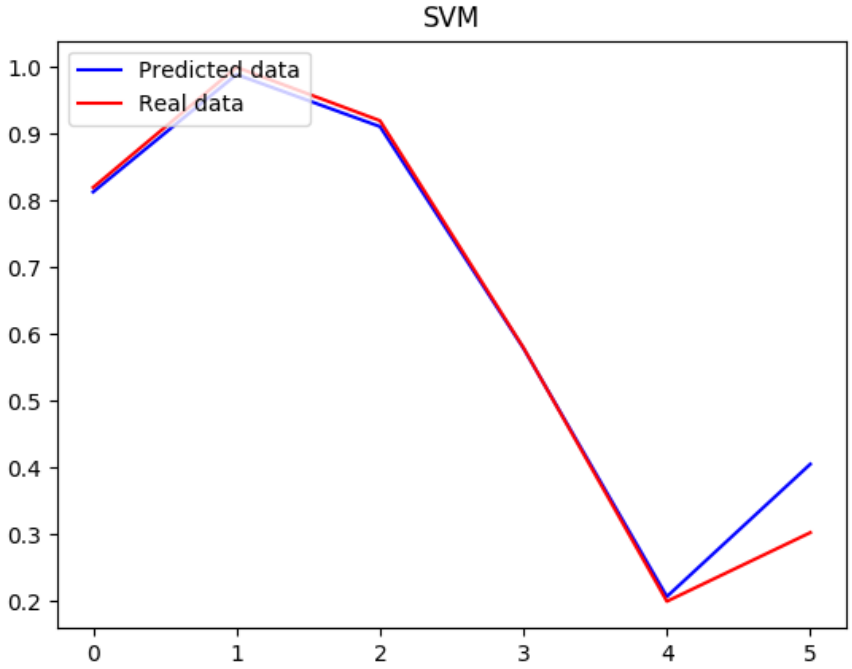

Figura 4. Resultats de la funció de testeig sobre una màquia de vectors suport ja entrenada amb unes dades de prova i un *kernel* lineal

Les dades que s'han subministrat al model anterior són tripletes *(a, b, c)* on tots els valors estan entre zero i ú. A la Figura 4, l'eix de les abscisses és el nombre de la dada dins del conjunt, i l'eix d'ordenades representa el resultat de predir c amb les *(a, b)* en la posició donada per la coordenada x.

5.3.5 Processador de dades

Com ja s'ha discutit abans, el pre-processament i el processament de les dades per a alimentar un model és un procés que té infinitat de variants, i que per tant, no és codificable en un programa en la forma d'una elecció, sinó que és necessari que l'usuari proporcioni aquestes funcions.

En el cas d'aquesta implementació de *Cronos*, que està feta en Python, hi ha la possibilitat de carregar una classe amb un cert nom des d'una altra[15]. Això fa que si nosaltres tenim un arxiu amb les funcions de pre-processament i processament i li comuniquem a la classe Sistema quin és el nom d'aquest fitxer, el sistema pot carregar aquests mètodes i utilitzar-los. L'únic que ha de complir aquest fitxer per a ser compatible amb la implementació actual és el següent:

- El fitxer ha de tenir una funció de pre-processament i una altra de processament.
- En el cas de les dues funcions l'entrada és una llista de llista de cadenes de caràcters (els fitxers llegits).
- En el cas d'ambdues funcions han de retornar una llista de llistes del tipus de variables que es necessitin segons el criteri de l'usuari.

En els annexos d'aquest treball i a mode d'exemples s'adjunta un processador de dades vàlid segons els requeriments descrits.

5.3.6 Programació de prediccions automàtiques

Desgraciadament aquesta part de l'entorn només és compatible amb les sistemes basats en Unix (MacOS, Linux), doncs es fa ús de la funcionalitat *Cron*[16] del sistema. Aquest component del sistema operatiu permet programar el llançament automàtic de les tasques que hi indiquem.

Per a fer automàtiques les prediccions, s'ha creat un script que comprova si s'ha afegit ja la tasca de fer prediccions en la *Crontab* (la llista de les feines que ha d'executar *Cron* periòdicament), i si no s'ha afegit la col·loca per a que s'executi cada 10 minuts. Al llançar-se l'execució automàtica d'aquesta tasca el que s'ha desenvolupat és la càrrega automàtica de tots els sistemes existents en el *Cronos* local i l'execució de la funció de predicció amb les dades d'entrada de tots aquells sistemes ja entrenats i amb la funció de predicció automàtica de valors activada. La sortida generada per cada model es desa en el fitxer de sortida indicat en cada sistema.

Cal afegir que encara que s'ha decidit executar aquesta tasca cada 10 minuts, és possible triar altres intervals de temps. Hi ha tota una sintaxi per al fitxer *Crontab* per a que *Cron* pugui saber cada quan llençar una tasca específica que l'usuari pot fer servir per a manipular l'execució de tasques segons les seves necessitats. Si es vol modificar, només caldria editar l'script *configurePredict.sh* creat juntament amb *Cronos*.

5.4 Validació

El procés de validació per a aquest entorn és doble:

- Per una part, està clar que hi poden haver *bugs* en el programa. Aquests errors han sigut detectats i solucionats a través d'usar el programa tant mentre es desenvolupava com posteriorment. S'ha fet una validació exhaustiva de cadascuna de les funcionalitats proposades en l'especificació del sistema provant-les.

- La validació de que les funcionalitats d'aprenentatge automàtic són correctes es realitzarà amb els dos casos d'estudi que en les seccions següents es descriuran de forma detallada (Cas A i Cas B).

6. Desenvolupament del Cas A

L'objectiu d'aquest cas és poder analitzar l'estat d'un núvol en quant a número de peticions de feina, i si aquest és capaç de mantenir el compliment del SLA en el futur pròxim o no. Per a tal fi es desenvoluparan dos parts diferenciades que després es podran combinar:

1 - Un algorisme que donat l'estat d'un núvol detecta si el servei està saturat.

2 - Un model de ML que predigui quina és la demanada estimada pel servei en un moment determinat. És important remarcar que aquest model ha de poder re-entrenar-se de forma fàcil a mesura que el *cloud* vaig generant més dades. Per tant el procés d'entrenament ha de ser fàcil i ràpid. Utilitzarem l'entorn *Cronos* per a construir el model d'aprenentatge automàtic.

La combinació d'aquests dos sistemes vindrà motivada pel fet de poder detectar si donada una nova demanda d'ús del servei en un instant t l'estat del núvol actual és capaç d'absorbir-la o no. També es podran recomanar accions a dur a terme, tant com per a solucionar l'escassetat de recursos, com l'abundància dels mateixos. En aquest context, engegar més servidors augmenta el nombre de recursos disponibles, i apagar-ne estalvia diners al servei.

6.1 Algorisme per a detectar la saturació del núvol

6.1.1 Anàlisi

El primer algorisme rellevant per al cas A és el de comprovar l'estat d'un núvol en un instant t donat. El que suposem per a aquest cas és el següent:

- tots els servidors que integren el *cloud* són el mateix model.
- el programa a executar és igual en tots ells.
- si correm el màxim nombre d'instàncies d'aquest programa en un ordinador del núvol qualsevol, aquest pot complir la tasca sense violar el SLA.
- si correm una instància més que el nombre màxim d'instàncies en un servidor, llavors s'incompleix el SLA per a tots els usuaris que estiguin executant feines en aquest ordinador.
- és l'usuari qui determina quan s'acaba l'execució del programa.

Amb aquestes premisses, és fàcil detectar si un núvol està saturat o no: és tant simple com mirar si existeix algun ordinador que pugui absorbir nova feina. Si existeix algun servidor que no estigui executant el màxim nombre de programes, llavors el núvol no està col·lapsat. En el cas de voler esbrinar si donada una quantitat de feina nova a tractar el *cloud* pot absorbir-la o no, cal mirar si és possible repartir aquesta càrrega de treball entre els servidors que tinguin encara espai per executar tasques noves.

6.1.2 Especificació

Aquest cas requereix de la implementació d'un algorisme que donada la càrrega de treball de cada ordinador d'un núvol, el màxim de feina que pot fer un servidor sense incomplir el SLA i el número de feines noves a executar-hi, tot respectant les restriccions i les premisses de la secció **6.1.1**, retorni si el servei és capaç de donar resposta a aquestes noves tasques i les anteriors o no.

Per tal de programar correctament aquest algorisme és necessari definir primer dos conceptes: el de **núvol estable** i el de **núvol òptim**. Donat que la representació que usarem d'un núvol és una llista que indica la càrrega de feina de cadascun dels seus servidors i un natural que indica el màxim de tasques pot fer cada ordinador, un núvol estable és un núvol que satisfà tota la demanda de feina. Un núvol òptim és aquell que satisfà tota la demanda de feina que té amb els recursos justos, sense malgastar-ne cap (per tant, tot núvol òptim és estable).

Les següents equacions determinen si un núvol donat està dins d'alguna de les dues categories o no:

$$Feina = Servidors \times FeinaPerSevidor$$

Equació 1. Equació dels núvols òptims

$$Feina \leq Servidors \times FeinaPerServidor$$

Equació 2. Equació dels núvols estables

Com es pot comprovar, és trivial veure que tots el núvols òptims són estables. Un núvol que no és estable és **inestable** i no és capaç de suportar tota la càrrega de treball que se li assigna, i per tant, necessita créixer en nombre de servidors.

6.1.3 Implementació

La implementació en Python d'aquest primer algorisme del Cas A és la següent:

```python
def can_cloud_accept_new_work(cloudState, work, workPerServer):
    for serverWorkload in cloudState:
        if serverWorkload < workPerServer:
            freeWorkspace = workPerServer - serverWorkload
            work = work - freeWorkspace
            if work <= 0:
                return True
    if work <= 0:
        return True
    else:
        return False
```

Figura 5. Implementació de l'algorisme per a detectar la saturació d'un núvol

CloudState és un a llista que, per cada ordinador *i* del núvol, conté un natural que indica quantes feines està duent a terme a la vegada. Aquest nombre mai és superior a *workPerServer*, que indica el màxim nombre de tasques que un servidor pot fer concurrentment, sinó s'incompliria el SLA. En addicció a les feines que ja s'estan executant, es vol processar *work* feines més. Tant *work* com *workPerServer* són hiper-paràmetres determinats per l'usuari.

L'algorisme retorna un booleà indicant si el núvol és capaç de processar també les noves feines o no. És a dir, l'algorisme retornarà *True* només si el conjunt de servidors formen o un núvol estable per a la càrrega de feina indicada en *work* i la que ja estaven executant prèviament.

Per últim, destacar que aquest algorisme es podria simplificar si la computació de la feina total que s'està fent el núvol es subministrés directament, però per a fer més explícit el seu funcionament s'ha volgut deixar expressament el paràmetre *cloud state* que indica quin és l'estat de cada servidor, ja que en algun lloc s'ha de realitzar el càlcul de la càrrega de feina i fer-ho dins de la funció és més explicatiu.

6.1.4 Validació. Raonament sobre la correctesa de l'algorisme

Donat a que aquest programa és molt simple, enraonaré la seva correctesa i després enumeraré els outputs per a alguns jocs de proves. El que volem esbrinar amb aquest programa és si existeix espai suficient en el nostre núvol com per a executar *work* tasques noves. Per tant, el que hem de fer és esbrinar quin és l'espai lliure en aquest; i si és superior o igual a *work*, significa que el nostre conjunt de servidors pot aguantar la nova demanda. Per aquesta raó, el que fem precisament és recórrer la llista que representa la feina que està executant cada servidor, esbrinar-ne l'espai lliure i tot seguit restar aquest espai lliure trobat al valor *work*, reassignant el nou valor a aquesta última variable. D'aquesta forma, si *work* finalment és igual o menor que zero, significa que el nostre servei pot satisfer la nova demanda (és un núvol estable).

A continuació es mostra una taula amb exemples d'entrades i sortides per a aquest algorisme:

Entrada	**Sortida**
`can_cloud_accept_new_work([],0,2)`	True
`can_cloud_accept_new_work([2,1],1,2)`	True
`can_cloud_accept_new_work([2,2],1,3)`	True
`can_cloud_accept_new_work([2,2,0,1],4,2)`	False

Taula 1. Taula amb diferents entrades i sortides per a l'algorisme del Cas A de detecció de saturació del núvol

6.2 Algorisme d'aprenentatge automàtic per a aproximar el nombre d'usuaris

6.2.1 Anàlisi

Si tenim un servei donat en el núvol i les seves dades d'ús, podem aproximar quin serà el nombre d'usuaris del mateix en un cert instant donat. Gràcies a l'aprenentatge automàtic, podem intentar trobar una correlació entre una data i el nombre d'usuaris del servei. Això ens permetria tenir un model que ens predigui quina serà la demanda estimada del servei en un moment determinat. Si tenim quin és el nombre d'usuaris que probablement vulguin fer ús del servei podem comprovar, de forma senzilla i per l'algorisme plantejat en la secció **6.1**, si el servei està preparat per a absorbir o no la demanda que es suposa que tindrà al cap d'un temps determinat.

També a mesura que passi el temps és possible que les condicions inicials del servei canviïn i les prediccions deixin de ser exactes. En aquest cas, hauríem de re-entrenar el model. Si aquest sistema es fa servir a producció interessa que aquest procés sigui el més ràpid possible, així que es buscarà un algorisme que s'entreni ràpidament.

6.2.2 Especificació. Dades per a entrenar el model de ML

6.2.2.1 Dades de Steam

Steam és la plataforma líder mundial de distribució de videojocs de forma digital. A part de ser una tenda, també ofereix suport per a poder jugar *online*. Per tant aquest servei té servidors habilitats per a jugadors i per a diferents jocs. Podem fer ús de les dades del nombre de jugadors de cada joc en una data determinada per a entrenar els models d'aprenentatge automàtic i així aproximar-ne la demanda.

Obtenció i format de les dades d'Steam

Les dades per a aquest procés es poden obtenir al web steamdb.info. Aquesta pàgina conté diferents mètriques relacionades amb Steam, i, concretament, informació sobre el nombre de jugadors d'un joc en un moment determinat. Aquesta web manté les dades del nombre de jugadors d'un joc durant la última setmana en intervals de 10 minuts, pel que la resolució de la informació és bona. Aquesta és descarregable des del mateix lloc web en diferents formats. Hi ha l'opció de descarregar les dades en *csv*, de forma que són fàcilment llegibles amb Python.

Cada document *csv* conté les següents dades en intervals de 10 minuts: data, nombre de jugadors, tendència de jugadors i espectadors en Twitch. La data és una cadena de caràcters en format estàndard fàcilment utilitzable. A partir d'aquesta transformació podem obtenir l'any, el mes, el dia del mes i la hora que corresponen a la cadena de caràcters inicial. El nombre de jugadors és un nombre natural que especifica quanta gent hi havia jugant en la data concreta al joc donat. La tendència de jugadors sembla ser la mitjana aritmètica d'un cert interval de temps del nombre de jugadors, pel que és un

nombre real. El nombre d'espectadors en Twitch és la quantitat de persones que estaven mirant retransmissions en línia del determinat joc a través de la plataforma Twitch (twitch.tv). Tot això ens proporciona molta informació fàcilment utilitzable per a entrenar algorismes de ML.

Les dades que utilitzarem són les de quatre jocs d'Steam: Civilization IV, Rocket League, The Witcher 3 i Borderlands 2. En el cas de validació de *Cronos* present podríem entendre les dades de cada joc com la representació del nombre d'usuaris d'una plataforma en el núvol.

6.2.2.2 No-linealitat de les dades

Abans d'entrenar diferents algorismes per a veure quins són millors en el cas present cal esbrinar si les dades recollides són lineals o no. En aquest cas, i tal com es pot observar en el següent gràfic, les dades del nombre d'usuaris dels jocs d'Steam no són lineals. De fet podríem aproximar cada dia a una corba gaussiana, on una setmana és una concatenació de gaussianes amb paràmetres diferents per cada dia (dades implícitament no lineals).

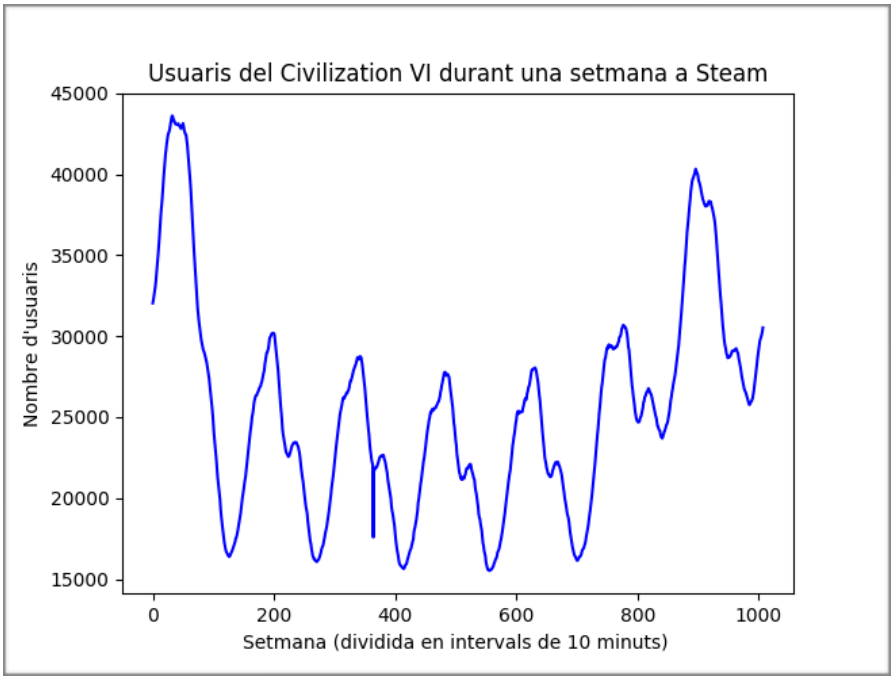

Figura 6. Nombre de jugadors mundial del *Civilization VI* durant una setmana a Steam

Donat a que les dades presenten aquesta característica és necessari fer ús de mètodes no-lineals de regressió per a poder predir amb el mínim d'error possible el nombre d'usuaris per una data donada.

6.2.3 Implementació

6.2.3.1 Pre-processament de les dades

Abans de crear dades per a entrenament i test, cal assegurar-ne la integritat de les mateixes. En el cas de les dades d'Steam, no sempre es sap el nombre d'usuaris en un moment determinat, doncs el servei pot estar en manteniment. Per tant, cal tractar les dades de nombre de jugadors buides. Una solució seria tractar-les com a zeros, però pot espatllar el model d'aprenentatge automàtic. Per tant, una bona idea és fer ús de les dades de tendència de jugadors, que són una aproximació temporal més realista que els zeros. Així, en la fase de pre-processament, substituirem els valors buits de "nombre de jugadors" pels de "tendència de jugadors" de la mateixa fila.

6.2.3.2 Processament de les dades

A l'hora de processar les dades de cada joc d'Steam que analitzem, cal ressaltar que les dades que s'utilitzaran són només dos:

- La data.

- El nombre de jugadors per a aquesta data determinada (que pot ser el mateix que la tendència de jugadors, com es pot veure en la secció de pre-processament de dades).

De la data hi ha la possibilitat d'extreure'n diverses característiques que ens permetran entrenar el model de ML:

- La hora i els minuts. En general, i tal com podem veure en la Figura 6, a unes hores determinades hi ha més jugadors que en altres; i solen ser hores similars.

- El dia de la setmana. Per exemple en el cap de setmana la gent té més temps lliure i per tant pot jugar més.

- El dia del mes. Pot ser que en determinats dies del mes hi hagi esdeveniments dins dels jocs que augmentin el nombre d'usuaris.

- El mes. Per un raonament similar al del dia de la setmana, a l'estiu la gent té vacances i per tant temps lliure. O a l'hivern la gent pot voler quedar-se a casa a causa del fred i aprofitar per jugar. Tots aquests factors que diferencien cada mes estan representats implícits en el nombre de jugadors de les dates que pertanyen a cada mes, si es que són rellevants.

- Any. Depenent de l'any hi pot haver una tendència alcista o baixista en el nombre de jugadors, així que està justificada la seva inclusió com a dada per al model.

Totes aquestes dades faran ús d'una pràctica molt comuna en ML: el *one hot encoding*. Aquesta tècnica el que fa es convertir una dada categòrica (com el dia de la setmana) en una llista binaria de tot zeros i només un valor a 1: el que correspongui al nombre del tipus determinat dins de la seva categoria. Això evita problemes del tipus que els diumenges (dia 6 de la setmana si el dilluns n'és el 0) tinguin més pes que els dilluns (dia 0). Així, per exemple, el *one hot encoding* per al dimarts (dia 1 de la setmana) seria [0, 1, 0, 0, 0, 0, 0]; i el del divendres (dia 4) seria [0, 0, 0, 0, 1, 0, 0].

6.2.3.3 Algorismes utilitzats

L'esquema d'alimentació dels models d'aprenentatge automàtic per a aquest Cas A amb les dades en el format explicat en la secció **6.2.3.2** és el següent:

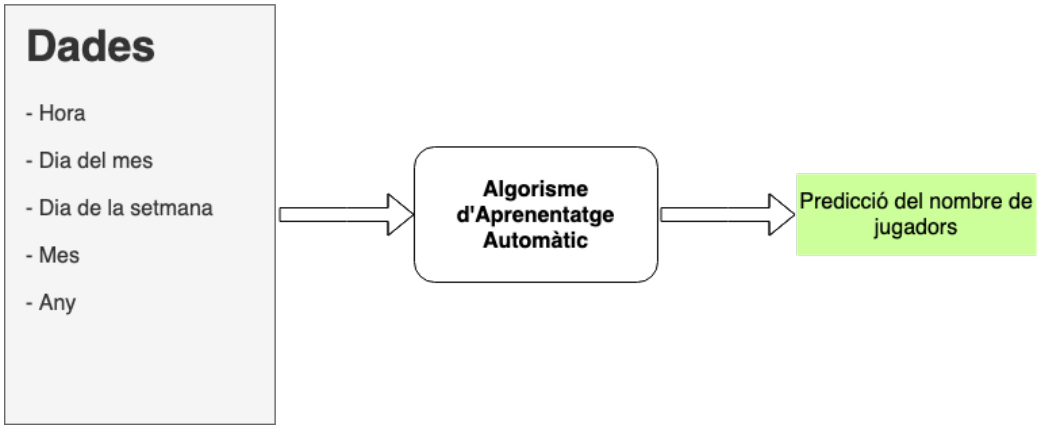

Figura 7. Esquema del procés d'alimentació dels models d'aprenentatge automàtic

Donada la naturalesa de les dades explicades en la secció **6.2.2.2** tots els algorismes utilitzats són mètodes no lineals. A continuació es mostra una taula amb els seus pros i les seves contres:

Mètode	**Pros**	**Contres**
Xarxa neuronal	- Pot representar relacions molt complexes en les dades gràcies a poder modificar la mida i el nombre de capes de neurones amagades.	- Com a més capes i neurones, més costa l'entrenament. - Cal executar diverses vegades l'entrenament per a evitar caure en un mínim local fàcilment superable. - Possibilitat d'*overfitting*. - Necessita moltes dades per entrenar-se i extreure'n tota la complexitat d'elles.

Mètode	Pros	Contres
Màquina de Vectors Suport (SVM)	- Més ràpida d'entrenar que la xarxa neuronal (en general). - Difícil fer-hi *overfitting.* - Amb poques dades d'entrenament s'obté bona precisió.	- El model pot acabar fent ús de molts vectors de suport.
Bosc aleatori (Random Forest)	- Més ràpida d'entrenar que la xarxa neuronal. - Amb poques dades d'entrenament s'obté bona precisió.	- Possibilitat d'*overfitting.*

Taula 2. Comparació entre els diferents algorismes no-lineals utilitzats per a modelar el problema

6.2.4 Validació

6.2.4.1 Objectiu inicial de precisió de la predicció a superar

La forma més naïf de calcular una aproximació al nombre d'usuaris en un instant t és agafar-ne les dades disponibles i fer-ne la mitjana. Per tant, el model elegit de ML ha de tenir un error menor al que presenti aquesta opció.

La precisió de l'aproximació a superar és molt baixa: el seu valor de R^2 és 0.0 (és a dir, no hi ha correlació entre les dades predites i les reals), i el MSE (*Minimum Squared Error*) és 44909.88. Aquests valors són una mitjana dels resultats del R^2 i el MSE per a cada joc.

6.2.4.2 Resultats dels diferents models

En la Taula 3 es mostra el rendiment entre els diferents algorismes presentats en la Taula 2. Els càlculs han estat fets tots en el mateix ordinador. Cada valor per a cada algorisme es calcula fent la mitjana ponderada del valor obtingut pel model en cada un dels quatre jocs dels que s'agafen dades. És important destacar que en aquesta taula a part de l'error en la predicció també es mostra el temps d'entrenament i predicció (ja que donats els requisits de rapidesa del cas en són crítics):

Algorisme	Hiper-paràmetres	Temps d'entrenament (s)	Temps de predicció (s)	R^2 mig	MSE mig
Model de l'objectiu a superar	N/A	$2x10^{-5}$	N/A	0,0	70.526,18
Xarxa neuronal	- Funció d'activació: tangent hiperbòlica - Learning rate: 0.01 - Entrenament amb descens de gradient optimitzat *adam* - Dues capes ocultes de 10 neurones cadascuna	11	0,05	0,76	12.744,2
Màquina de Vectors Suport (SVM)	- C = 10000 - ε = 0.0001 - Kernel RBF	0,84	0,37	-0.49	43.424,30
Bosc aleatori (Random Forest)	- Nombre d'arbres = 100	1,27	0,04	0,19	25.297,49

Taula 3. Comparació de rendiment dels diferents models d'aprenentatge automàtic

En la Taula 3 es mostren els hiper-paràmetres de cada model usats per a configurar-lo, el temps d'entrenament que es triga amb les dades i els hiper-paràmetres elegits, el temps de predicció al voler calcular una nova predicció amb unes dades determinades, el coeficient de relació de les dades d'entrenament i validació R^2 i l'error (*Mean Squared Error, MSE*) mig.

Com es pot observar en la taula superior, es supera fàcilment l'objectiu inicial de precisió a millorar de la secció **6.2.4.1**, on tots els algorismes utilitzats aconsegueixen resultats superiors. En el cas del temps gastat en les xarxes neuronals, veiem que comparativament és molt superior al dels altres algorismes, ja que triga onze segons a entrenar-se. Això és perquè realitzem diferents entrenaments[17] (en aquest cas, 10) per a quedar-nos amb el millor, doncs les xarxes neuronals tenen la particularitat de que en l'entrenament per descens de gradient es pot trobar un òptim local i no un absolut. Comentar però que les xarxes neuronals són una mica complexes d'utilitzar perquè per a l'entrenament el nombre de jugadors s'ha de normalitzar entre 0 i 1 i fer la transformació posterior de la predicció per a treure'n el valor i no deixar-ho en aquest rang.

Amb tot i superar l'objectiu inicial, el rendiment dels mètodes usats no és suficientment bo. Amb aquesta motivació podem fer canvis, especialment en el que respecta a la forma en que processem les dades per a l'entrenament i la validació, per a intentar aconseguir uns resultats superiors.

6.2.5 Canvis en el processament de dades. Nova validació

Amb l'objectiu de millorar el rendiment del sistema proposat en aquest apartat, es pot millorar el model de dades proporcionat per a l'entrenament i la fase de test dels algorismes. Fins ara, consideràvem que cada dada era independent de les anteriors, però si adoptem la idea de que un valor és una funció de, com a mínim en part, els seus valors anteriors, obtenim una sèrie temporal.

Sigui Δt el temps entre predicció i predicció del nombre d'usuaris; fins ara per a predir el nombre d'usuaris del servei en $t+\Delta t$ fèiem ús de la data de l'instant $t+\Delta t$, però també es pot fer ús el valor del nombre de jugadors real en t per a intentar predir aquest nombre. Per tant, en l'instant t intentem predir el nombre de jugadors en $t+\Delta t$ fent ús de la data de $t+\Delta t$ i del nombre de jugadors reals en t. Així, el valor $t+\Delta t$ en depèn de la data associada a ell i de l'anterior valor en t.

El canvi en el model de dades és factible perquè cada cert període de temps Δt sabem amb exactitud quin és el nombre de jugadors d'aquell moment, doncs són dades que podem obtenir. I així, podem usar-les per a predir la demanda d'aquí a aquest cert temps. El següent gràfic il·lustra el nou model:

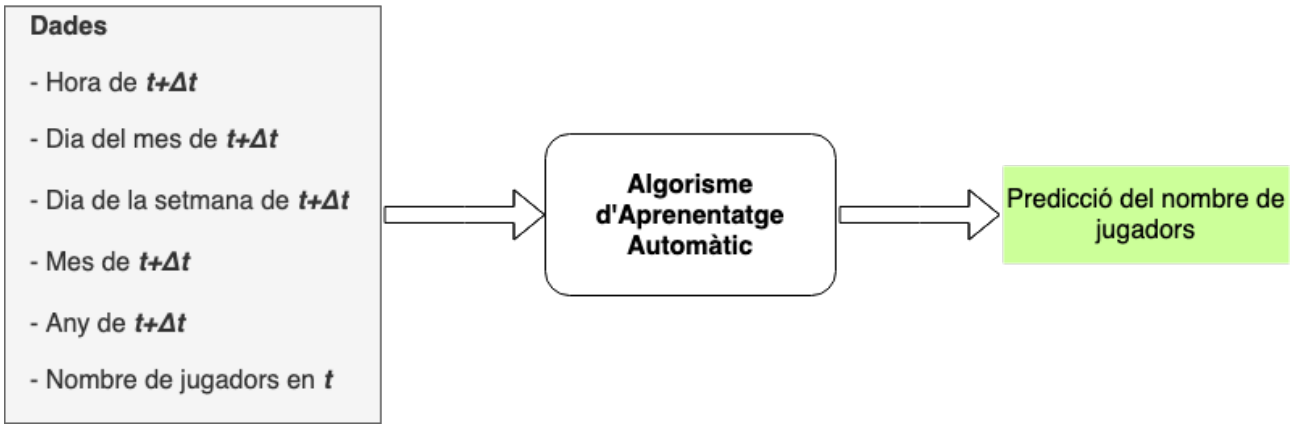

Figura 8. Esquema del procés d'alimentació dels models d'aprenentatge automàtic amb el canvi en el model de dades

6.2.5.1 Rendiment de *Random Forest* amb les noves dades

Donat a que l'algorisme de Boscs Aleatoris no requereix d'una normalització de les dades, podem alimentar aquest model amb les noves dades directament, ja que com hem comprovat abans les dates en *hot encoding* és capaç d'entendre-les; i els nombres enters

(com és el cas del nombre de jugadors) també. Donat a que les dades que tenim de *test* i *trainning* són les descarregades d'Steam, per a construir el nou model el que hem de fer és per a cada fila de dades afegir-hi també el resultat d'usuaris *online* de la fila immediatament anterior. En la següent taula se'n mostra el nou rendiment:

Algorisme	**Hiper-paràmetres**	**Temps d'entrenament (s)**	**Temps de predicció (s)**	**R^2 mig**	**MSE mig**
Bosc Aleatori (Random Forest)	- Nombre d'arbres = 10	0,17	0,018	0,99	160,42

Taula 4. Rendiment de *Random Forest* amb el nou format de dades

La precisió de l'entrenament amb les noves dades amb l'algorisme de *Random Forest* és molt alta. Per tant, aquest és un molt bon model, que s'entrena i que prediu ràpid: és un algorisme que ens serveix per als propòsits d'aquest apartat i que en compleix tots els requisits. Per suposat, aquesta precisió millora per molt les mètriques de rendiment obtingudes a la Taula 3 (on, per exemple, el MSE més baix era 12.744,2).

Al veure que el rendiment amb aquest model és tant alt i eficient, s'ha pres la decisió de quedar-se amb aquest.

6.3 Acoblament del subsistemes del Cas A

Un cop tenim construïts i validats l'algorisme de detecció de la saturació del núvol (secció **6.1**) i el model d'aprenentatge automàtic per a aproximar el nombre d'usuaris (secció **6.2**), els podem ajuntar per a poder treure més profit dels dos i satisfer els objectius proposats per a aquest Cas A. La unió d'aquests dos mòduls ens habilitarà les següents possibilitats:

- Predir amb gran precisió quants usuaris nous tindrem al cap d'un temps Δt, i comprovar si el servei està en condicions de satisfer la nova demanda o no.
- Recomanar encendre o apagar servidors, ja sigui per a donar servei a més usuaris o per a estalviar costos.

De forma intuïtiva, el que hem de procurar de fer és predir el nombre d'usuaris en un moment determinat, mirar l'estat del núvol i el seu nombre d'usuaris real, comparar-ho amb aquest valor predit i recomanar engegar o apagar servidors (assumint que el valor que endevinem és sempre correcte). Això ens permetria sempre tenir un núvol òptim, tal com es va definir en l'**equació 1**.

Aquesta solució és correcta assumint que tots els actors implicats en el procés són ideals. Un dels punts de generació de problemes més grans que ens podem trobar (i que podem resoldre) és el que va relacionat amb el temps d'engegar nous servidors. Concretament, el

problema més gran està en no poder assegurar el compliment del SLA perquè no tenim els suficients servidors per a executar les feines demanades: si el temps d'engegar nous servidors és superior al període en que anem fent noves consultes al model de ML per a saber el nombre de jugadors aproximat al cap de Δt, es podria donar el cas maliciós de que prediguem una baixada del nivell de treball i en conseqüència apaguem servidors, només per veure immediatament després que hi ha una nova pujada de la demanada però no tinguem el temps suficient com per a engegar tots els servidors necessaris per a atendre les noves peticions i complir el SLA.

En la següent imatge es pot apreciar la problemàtica exposada. Quan hi ha menys servidors que usuaris, el servei deixa de complir el SLA. Això és degut a que hi ha un temps d'engegar servidors comparativament més gran que les fluctuacions del nombre d'usuaris. Per tant, quan el nombre de servidors (línia blava) està per sota de la línia vermella, s'està incomplint el SLA perquè no tots els usuaris poden ser atesos:

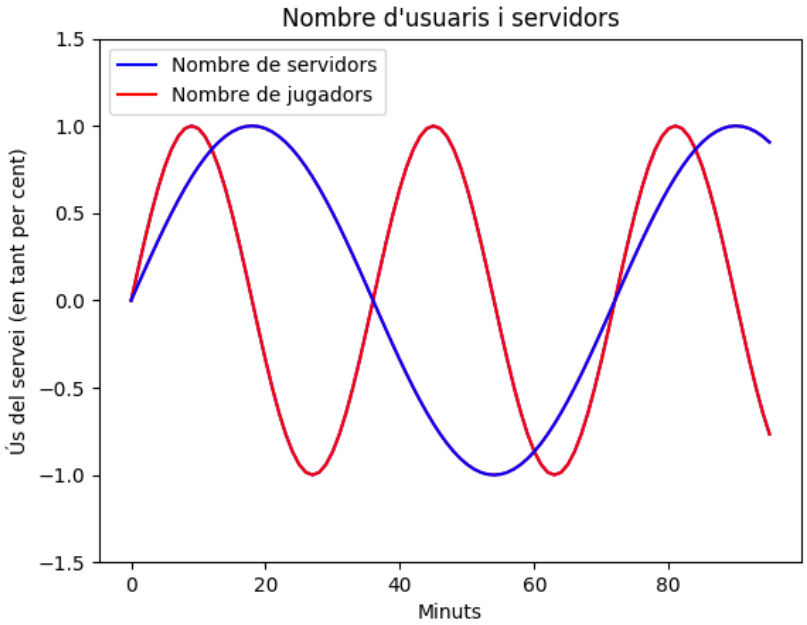

Figura 9. Gràfica del nombre de jugadors i de servidors

Una solució a aquesta problemàtica és la implementada en aquest treball. La idea és mirar quants períodes de predicció (Δt) caben dins del temps mig d'engegar nous servidors, i mirar quin és el màxim de feina predit per a aquest interval de temps. Així, encara que el núvol no sigui òptim, sí que serà estable (**equació 2**) i complirem el SLA. Per tant, l'algorisme assegura que si l'estat del núvol de l'entrada és estable, també serà el proposat per la sortida.

Finalment, s'ha modificat l'algorisme de la primera part del Cas A d'acord amb l'explicat en aquesta secció. Els dos programes s'han pogut acoblar, i així queda demostrat que és possible fer servir l'entorn *Cronos* amb altres programes externs que en facin servir les seves prediccions com a informació clau per a generar recomanacions, accions, etc.

7. Desenvolupament del Cas B

7.1 Fase d'anàlisi

En el Cas B el que volem és construir un model d'aprenentatge automàtic que ens digui quin és el percentatge de cada programa en un moment determinat que s'executa respecte al total. Això ens permetrà assignar amb previsió i de forma més exacte els recursos necessaris per al futur immediat del nostre núvol, assegurant un bon compliment del SLA i un equilibrat gast de recursos (tant informàtics com financers), ja que amb aquesta informació es poden dur a terme diferents accions per a assegurar el bon funcionament del nostre núvol.

Com a exemple pràctic, si tenim un núvol que pot executar programes de tipus ***A***, ***B*** i ***C***, el que es pretén és trobar quin és el percentatge de cada un d'ells en un instant concret. Així, un cop executat el nou mòdul, podríem saber que un 20% de programes seràn del tipus ***A***, un 43% del tipus ***B*** i un 37% del ***C***.

A partir d'unes dades d'entrenament aquest sistema ha de ser capaç de predir per a un instant donat quin seria aquest percentatge per cada programa. Per tant, és evident que fer ús d'algorismes d'aprenentatge automàtic amb una fase d'entrenament i una de validació s'ajusta a les nostres necessitats, i que podem crear aquest model amb l'entorn *Cronos*.

7.2 Especificació

Per a entrenar el model d'aquest cas, farem ús de les mateixes dades del Cas A (apartat **6.2.2.1** d'aquest document). Per tant, i tal com s'explica en la secció immediatament anterior mencionada, aquesta informació compleix totes les característiques ja explicades (com per exemple que aquestes dades són no-lineals). El nostre sistema de predicció de probabilitats diferenciarà doncs entre els quatre jocs dels que tenim dades: el Civilization IV, el Bordelands 2, el Rocket League i el The Witcher 3. Cal recordar que tal com s'explica amb anterioritat, les dades són no-lineals.

7.3 Implementació

7.3.1 Pre-processament de les dades

El pre-processament d'aquestes dades és anàleg al ja explicitat en la secció **6.2.3.1**.

7.3.2 Processament

Un cop assegurada la integritat de les dades gràcies al pre-processament, cal processar la informació per tal d'obtenir els tant per cent de llançaments de cada joc dels estudiats de Steam, per tal d'obtenir un conjunt de dades on cada fila tindria:

- **Una data**. La data es processa i s'utilitza igual que en el cas A, tal com s'explica en el punt anterior **6.2.3.2**.

- **Un percentatge del llançament d'un joc respecte al total, per a cada joc**. Per cada fila, el conjunt dels percentatges ha de sumar 1.

7.3.2.1 Procediment per a obtenir les probabilitats de cada joc

Per tal d'obtenir aquest resultat, el primer que fem és obtenir totes les dades de forma que cada fila contingui una data i el nombre de jugadors d'un joc determinat per a aquella data (i per cada joc). El format d'aquestes dades es pot entendre millor en una taula:

Data	Civilization VI	Rocket League	Borderlands 2	The Witcher 3
t	32217	26978	36409	13751
t + 10 minuts	32457	27468	36777	13911
t + 20 minuts	32669	27864	37311	13987

Taula 5. Nombre de jugadors per joc

Arribats en aquest punt, pot semblar temptador canviar l'objectiu d'aquest cas intentant obtenir la probabilitat de que un llançament d'un nou joc correspongui a un de determinat. El problema que tindria aquest objectiu és que amb les dades disponibles per a aquest apartat només podem obtenir la diferència entre la gent que decideix executar un nou programa i els que el paren. Així, podríem saber que la diferència entre jugadors d'un joc és 0, però aquesta dada pot amagar que 1.000 persones han començat a jugar i 1.000 més han parat. Per a tenir unes bones prediccions sobre aquest fet necessitaríem les dades de les persones que comencen i les que acaben de jugar per separat.

L'últim pas consisteix en transformar cada fila de manera que els valors dels jocs sumin en total 1. Per a realitzar tal objectiu, és suficient amb sumar tots els valors dels jocs de la fila i dividir cada nombre per aquest total. D'aquesta forma ja tindrem la probabilitat de que un nou llançament de joc pertanyi a un joc determinat en un instant donat. Si la suma resulta ser 0, assignarem a totes les probabilitats zeros. La taula que mostra el resultat (en tant per cent, no sobre 1) de transformar les dades de la taula superior amb el procediment descrit en aquest paràgraf és la següent:

Data	Civilization VI (%)	Rocket League (%)	Borderlands 2 (%)	The Witcher 3 (%)
t	15,57	42,42	30,55	11,44
t + 10 minuts	19,07	38,95	29,25	12,71
t + 20 minuts	17,40	32,51	43,84	6,24

Taula 6. Taula amb les probabilitats de cada joc

En la taula anterior i donat un instant t, el que significa cada nombre associat a cada joc és la probabilitat de que una execució d'un programa sigui d'aquest en concret. Fer notar que entre cada fila de dades ja transformades passen 10 minuts, doncs les dades processades són les que provenen d'Steam.

7.3.3 Algorismes utilitzats

Per al Cas B utilitzarem els mateixos algorismes no lineals que en el cas A: xarxes neuronals, boscs aleatoris i màquines de vectors suport. Els punts forts i fluixos de cada mètode s'expliquen en la taula 2 que hi ha a la secció **6.2.3.3**.

En quant a la forma d'alimentar els diferents algorismes, al principi s'ha intentat fer de la següent manera:

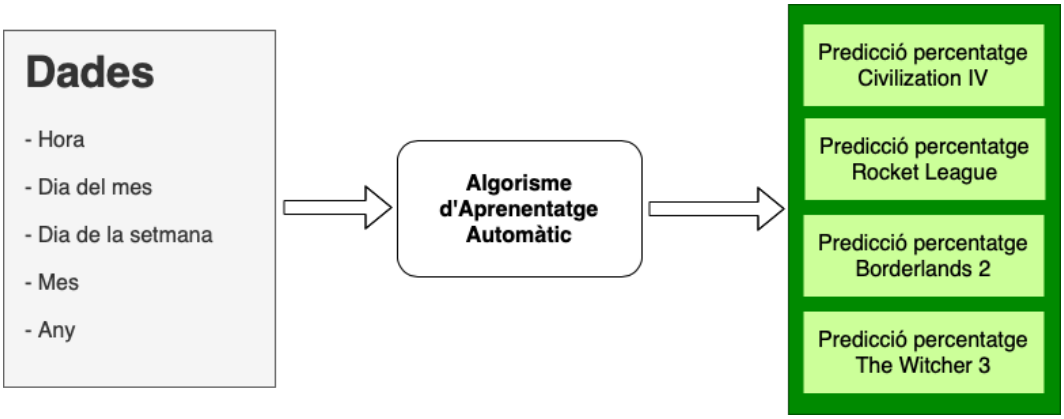

Figura 10. Procés inicial d'alimentació dels diferents algorismes del Cas B

Però s'ha vist que el rendiment és molt dolent per a tots els algorismes en les primeres proves fetes. Per tant, ràpidament s'ha canviat a en contes de tenir un model que ens retorni cada una de les probabilitats, tenir un model entrenat per a cada joc. D'aquesta forma, separant les diferents feines, obtenim molt millor rendiment. El resultat és el següent:

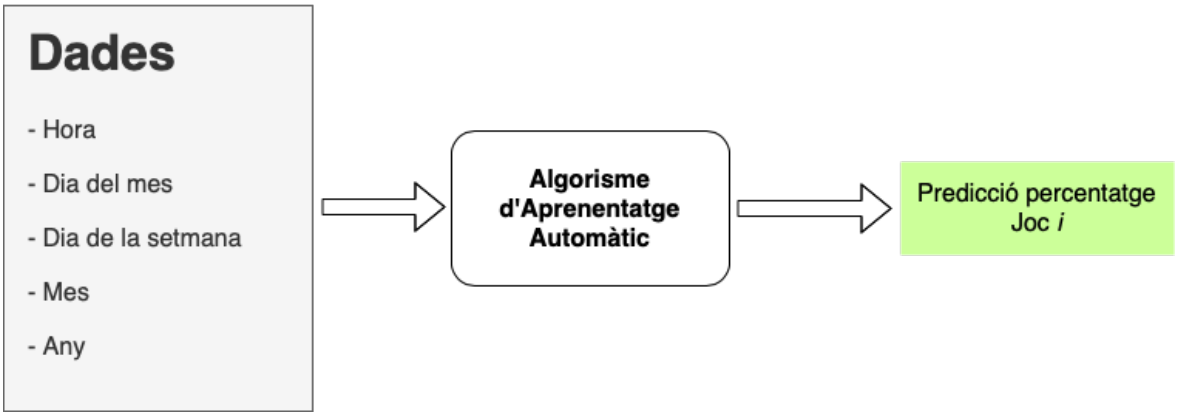

Figura 11. Procés definitiu d'alimentació dels diferents algorismes del Cas B

7.4 Validació

7.4.1 Objectiu inicial de precisió de la predicció a superar

En aquest cas per a establir un model simple a partir del qual comparar els demés una opció fàcil es calcular quina és la mitjana total del tant per cent d'un joc i mirar quin és l'error que té en total aquesta aproximació. Per tant, el model elegit de ML ha de tenir a la força un error menor al que presenti aquesta opció, ja que aquesta primera aproximació no conté cap model d'aprenentatge automàtic.

En la taula següent es mostra el coeficient R^2, el MSE (Mean Square Error) i el MAE[3] (Mean Absolute Error) per a cada joc del *dataset* d'Steam amb l'algorisme explicat en el paràgraf anterior:

	R^2	**MSE**	**MAE**
Civilization VI	-0,4	0,0011	0,027
Borderlands 2	-16,63	0,0055	0,071
Rocket League	-0,83	0,0073	0,0691
The Witcher 3	-0,01	0,0013	0,0335

Taula 7. Rendiment de l'algorisme ingenu per a predir quotes d'un programa

7.4.2 Resultats dels diferents models

A continuació es mostraran els resultats de predir el tant per cent de quota d'un programa en un moment donat per als quatre jocs analitzats. Per a cadascun dels programes, es mostra una taula separada amb els mateixos indicadors que la taula 8. Un cop mostrades totes les taules es procedeix a analitzar quin algorisme és el millor, segons els nostres criteris i requeriments. Els càlculs han estat fets tots en el mateix ordinador.

7.4.2.1 Resultat per al Civilization VI

	Configuració	**R^2**	**MSE**	**MAE**
Algorisme naïf	-	-0,4	0,0011	0,027

[3] **MAE (Mean Absolute Error):** mesura de l'error basada en fer una mitja dels valors absolut de la diferència entre el valor real i el predit de cada valor d'un conjunt de dades. Útil quan les diferències són valors entre zero i ú, perquè amb aquests valors el MSE al elevar-los al quadrat els fa més petits.

	Configuració	R^2	MSE	MAE
Xarxes neuronals	- Funció d'activació: logística - Solver: adam - Learning rate: 0.01 - Una capa oculta de 20 neurones	0,31	$5{,}32 \times 10^{-07}$	0,0192083
Random Forest	- Nombre d'arbres = 10	0,09	$7{,}11 \times 10^{-07}$	0,0224
SVM	- C = 10000 - ε = 0.01 - Kernel RBF	0,33	$5{,}25 \times 10^{-07}$	0,019

Taula 8. Comparació del rendiment per als diferents models amb les dades del *Civilization VI*

7.4.2.2 Resultat per al Rocket League

	Configuració	R^2	MSE	MAE
Algorisme ingenu	-	-0,83	0,0073	0,0691
Xarxes neuronals	- Funció d'activació: logística - Solver: adam - Learning rate: 0.01 - Una capa oculta de 10 neurones	0,53	$1{,}85 \times 10^{-06}$	0,0369
Random Forest	- Nombre d'arbres = 10	-1,56	$1{,}0117 \times 10^{-05}$	0,0978
SVM	- C = 10000 - ε = 0.01 - Kernel RBF	0,14	$3{,}3764 \times 10^{-06}$	0,053

Taula 9. Comparació del rendiment per als diferents models amb les dades del *Rocket League*

7.4.2.3 Resultat per al Borderlands 2

	Configuració	R^2	MSE	MAE
Algorisme ingenu	-	-16,63	0,0055	0,071
Xarxes neuronals	- Funció d'activació: logística - Solver: adam - Learning rate: 0.01 - Una capa oculta de 10 neurones	-1,58	$7{,}926 \times 10^{-07}$	0,0233
Random Forest	- Nombre d'arbres = 10	-1,53	$7{,}7746 \times 10^{-07}$	0,0264
SVM	- C = 10000 - $\varepsilon = 0.01$ - Kernel RBF	-16,19	$5{,}2707 \times 10^{-06}$	0,0681

Taula 10. Comparació del rendiment per als diferents models amb les dades del *Borderlands 2*

7.4.2.4 Resultat per al The Witcher 3

	Configuració	R^2	MSE	MAE
Algorisme ingenu	-	-0,01	0,0013	0,0335
Xarxes neuronals	- Funció d'activació: logística - Solver: adam - Learning rate: 0.01 - Dues capes ocultes de 10 neurones cadascuna	0,92	$1{,}03 \times 10^{-07}$	0,007693
Random Forest	- Nombre d'arbres = 10	0,91	$1{,}207 \times 10^{-07}$	0,0092
SVM	- C = 10000 - $\varepsilon = 0.01$ - Kernel RBF	0,92	$1{,}091 \times 10^{-07}$	0,0082

Taula 11. Comparació del rendiment per als diferents models amb les dades del *The Witcher 3*

7.4.2.5 Conclusions

Com es pot observar en les taules superiors, en cadascun dels quatre jocs el model que té menys error absolut mitjà és la xarxa neuronal (degudament configurada per a cada cas). A més, supera les expectatives col·locades per l'algorisme ingenu. El problema és que per a alguns jocs els resultats són molt bons, però per a altres no. A continuació es mostren dos gràfics amb prediccions fetes per les xarxes neuronals, amb les dades del *Borderlands 2* i amb les del *The Witcher 3*, respectivament:

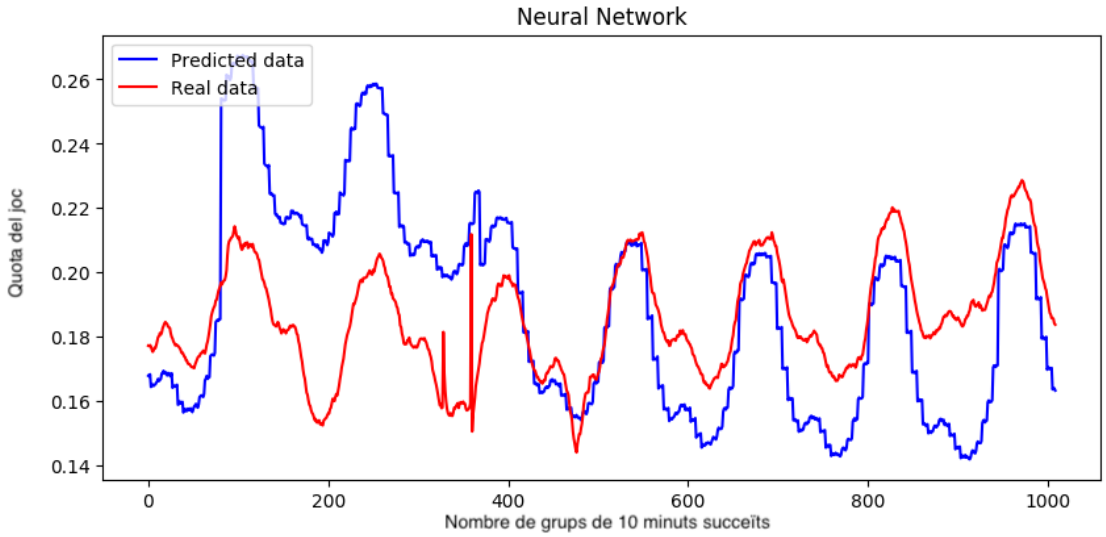

Figura 12. Predicció de la quota per al *Borderlands 2*

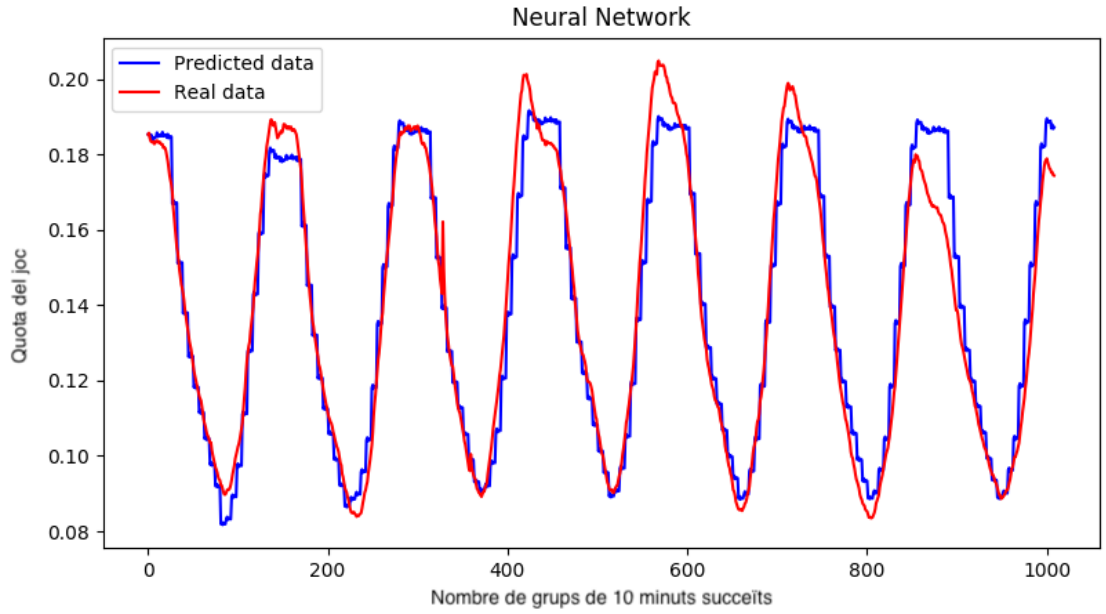

Figura 13. Predicció de la quota per al *The Witcher 3*

Observant les Figures **12** i **13** és evident que encara que per a certs jocs la solució és bona, per a altres encara hi ha marge de millora. En el cas del *Borderlands 2* és molt clar que una predicció com les que es fa en els primers valors és inacceptable per a un bon model.

7.4.3 Canvis en el processament de dades. Nova validació

Per a intentar reduir l'error en els models d'aprenentatge automàtic ja explicats, el que farem és adoptar la mateixa tècnica que en el Cas A: com que aquest sistema que estem creant es suposa que anirà predient dades a mesura que passi el temps de forma continuada, podem fer ús de les dades reals anteriors per a subministrar més informació al model. Per a realitzar aquest canvi, l'única cosa que s'ha de fer és modificar la funció de *process* del model B dins de l'entorn integral de ML desenvolupat *Cronos*. En contes de subministrar només la data al model, l'alimentarem també amb el valor de la quota sobre el tant per cent total d'un joc determinat.

Per tant, i de forma molt similar a l'apartat **6.2.5**, el sistema es pot resumir en el següent esquema:

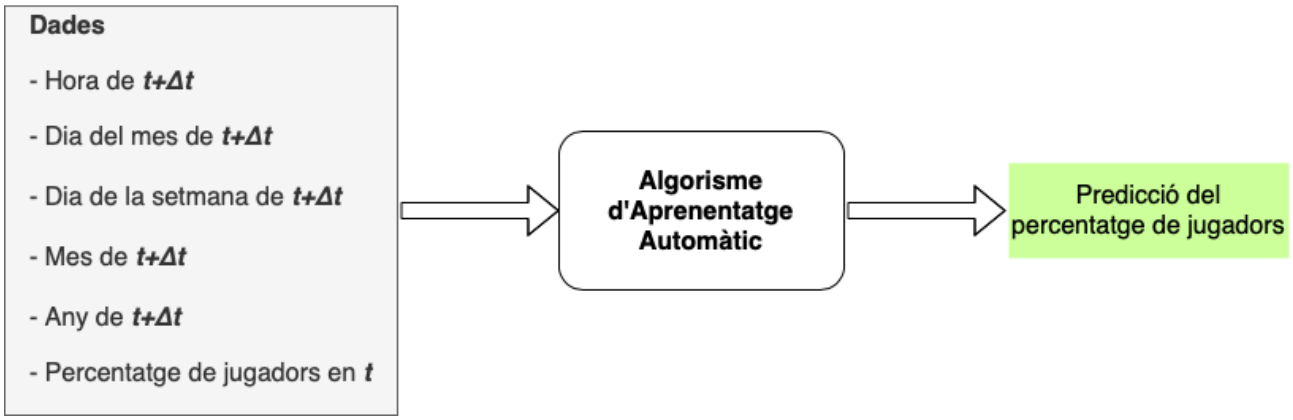

Figura 14. Esquema del procés d'alimentació dels models d'aprenentatge automàtic amb el canvi en el model de dades

7.4.3.1 Nova validació

Gràcies al canvi en el processament de les dades, hem pogut millorar una mica l'exactitud de les xarxes neuronals dels diferents jocs. A continuació es mostra una taula amb la comparació dels vells i els nous *Mean Absolute Error:*

	MAE vell	**MAE nou**
Civilization VI	0.0192083	0.011854582857433954
Rocket League	0.0369	0.02311483270681685
Borderlands 2	0.0233	0.02151028205811475
The Witcher 3	0,007693	0.006189098819714625

Taula 11. Nous resultats per a les xarxes neuronals amb el canvi en el processat

Com es pot observar, hem obtingut millores en el MAE de totes les xarxes neuronals, però el *Borderlands 2* no s'ha reduït suficient l'error. En aquest punt, és interessant comparar l'error del *Rocket League* i d'aquest joc: encara que el MAE del *Rocket League* és més alt, els resultats són més consistents (ja que es segueix produint l'error que es pot observar en la Figura 12 al predir dades del *Borderlands 2*).

Per a demostrar les capacitats de Cronos s'ha volgut deixar expressament com a definitiu el model de les xarxes neuronals per al Cas B, doncs un model basat en *Random Forests* també tenia molt bon rendiment en aquest cas (un altre cop).

8. Acoblament del Cas A i el Cas B

Un cop tenim ja els dos casos preparats cobra molt de sentit retroalimentar un cas amb la informació de l'altre. Donat a que el cas B intenta donar quina és la probabilitat de que una execució sigui d'un programa en concret, el cas A pot fer ús d'aquesta informació per a saber quin és el nombre d'instàncies de cada programa que es volen executar.

Gràcies a l'acoblament podem eliminar una restricció del Cas A: no tots els programes executats seran el mateix. De totes formes, i en falta d'una bona manera de "pesar" els programes, seguim assumint que tots els programes requereixen dels mateixos recursos.

L'algorisme per usar en conjunt aquests dos models és senzill. Donat una data t i un valor j' que correspon al nombre de jugador en $t - 1$:

- Predir el nombre d'usuaris j del servei per a t sabent que en $t - 1$ n'hi havia j', gràcies al Cas A.
- Calculem per a tot programa del que es vulgui saber la quota quina és la probabilitat p_i de que una instància d'una feina en t correspongui a aquest programa, gràcies al Cas B.
- Per cada probabilitat p_i calculada:
  - Multiplicar $j \cdot p_i$. El resultat és el nombre d'usuaris que predeim que estaran executant aquest programa.

Cal afegir que en cas de que es vulgui calcular la probabilitat de cadascun dels programes que es poden executar en un servei en el núvol, un cop s'han predit totes les probabilitats és bona idea normalitzar-les de tal forma que la suma de totes les quotes sigui 1. Sinó, al voler calcular el total d'usuaris de cada programa, la suma pot ser més gran que el nombre d'usuaris predits pel Cas A.

L'acoblament entre els dos casos s'ha pogut realitzar gràcies als sistemes d'aprenentatge automàtic ja entrenats dins de l'entorn *Cronos* i un programa extern a aquest, creat per a l'ocasió, que en gestiona la coordinació dels dos models, les dades d'entrada i les dades de sortida.

9. Planificació

9.1 Planificació inicial

Aquest projecte té com a data d'inici el dia 1 de març de 2019 i té una previsió de ser acabat a principis de juny, ja que la dedicació orientativa que recomana la FIB és de 540 hores (donat a que són 18 ECTS i cada ECTS és, aproximadament, 30 hores de treball). Si es treballa 8 hores diàries en el projecte, hi ha marge per a acabar-lo i abordar tots els inconvenients que hi puguin aparèixer en el seu desenvolupament.

9.1.1 Identificació de tasques. Objectius.

El projecte, com es va explicar ja a la secció **1.3** d'objectius, pretén dur a terme l'anàlisi, el disseny, la implementació i la validació d'un sistema d'aprenentatge automàtic configurable i extensible que detecti o prevegi violacions del SLA d'un servei en el núvol. Per tant, és raonable dividir les tasques en aquestes seccions que configuren el desenvolupament de la solució volguda.

9.1.1.1 Aprenentatge

La fase prèvia a la creació del software ha sigut per a informar-se i aprendre sobre les eines i tecnologies a fer servir. Aquesta fase ha sigut la primera per a poder agafar una visió àmplia sobre les tècniques a usar. En concret, s'ha estudiat com funciona i com extreure dades del servei *Elastic Search* [18], com processar dades per a entrenar models de *machine learning* (també tenint en compte el factor temps); quins algorismes de ML fer servir (un altre cop, sabent que s'ha de tractar el temps) i s'ha estudiat quina és la millor forma actual de crear un software portable per al projecte.

Cal remarcar que encara que aquesta fase té un temps delimitat al principi del projecte, durant el desenvolupament d'aquest he anat revisant contínuament informació útil per a al projecte; sobretot en l'àmbit del desenvolupament en Python (instal·lació i ús de llibreries externes, E/S a fitxers, programació de crides periòdiques a funcions a través del sistema...).

9.1.1.2 Anàlisi

En la fase d'anàlisi s'ha parlat amb la empresa Ludium Lab per a intentar aconseguir una idea general del projecte sense que sigui específica per a aquesta empresa. L'objectiu a aconseguir era desacoblar la dependència de l'estructura de dades d'aquesta per a que qualsevol servei basat en el núvol pugui utilitzar l'entorn desenvolupat.

9.1.1.3 Disseny

En la fase de disseny, podem distingir tres tasques dutes a terme:

- Especificació del sistema: es tracta d'especificar l'arquitectura general de l'entorn a desenvolupar.
- Especificació del format de les dades: l'objectiu d'aquesta tasca ha sigut trobar quines dades i com es necessiten per a que el sistema les pugui consumir, amb la

fi d'entrenar algorismes de ML i després fer comprovacions sobre les violacions del SLA.

- Especificació de les funcions externes a l'entorn per al Cas A i el Cas B: encara que l'entorn desenvolupat compleixi totes les tasques relacionades amb l'aprenentatge automàtic, hi ha codi extern (sobre tot relacionat amb l'*output* de l'entorn) per a cada cas que és necessari especificar.

9.1.1.4 Implementació

En aquest apartat s'enumeren totes les tasques relacionades amb la part de programació del projecte, a més de les d'obtenir dades. Són les següents:

Tasques de recollida de dades:

- Trobar i recollir dades de Steam per a diferents joc.
- Visualitzar les dades obtingudes. Aquest pas és important doncs visualitzar les dades ens permet modelar-les després de forma encertada.

Tasques destinades a implementar el software:

- Implementar una interfície d'ús de l'entorn via comandes.
- Implementar codi per a poder connectar les fonts de dades amb el programa segons el format especificat en la fase **10.1.3** de Disseny.
- Implementar el processament de dades per a preparar-les per a l'entrenament dels models de ML.
- Implementar l'entrenament dels diferents algorismes d'aprenentatge automàtic.
- Implementar codi per a veure quins algorismes són millors.
- Implementar una forma de predir periòdicament amb cada sistema ja entrenat i que es vulgui predir, donat que hi hagi noves dades per a aquest sistema dins l'entorn.
- Implementar un sistema per a guardar l'estat dels diferents models d'aprenentatge automàtic que tinguem en l'entorn.

9.1.1.5 Validació del sistema

Com ja es va explicar en l'apartat **4.2** de Metodologia de validació, hi ha dos casos que gràcies a les dades de Steam i el sistema creat podem comprovar. A través d'aquesta informació es pot veure si la implementació feta tant de l'entorn com del codi específic de cada cas compleix els nostres objectius, tant de funcionalitat com de rendiment i minimització de l'error.

9.1.1.6 Tasques extres

Aquestes tasques no tenen dependències amb les d'obtenció i processament de dades o les d'aprenentatge automàtic. Són:

- Dissenyar i implementar la interfície gràfica del programa.
- Dissenyar i implementar un sistema de notificacions.

9.1.1.7 Creació de la documentació

Amb la finalitat de que qualsevol organització interessada pugui fer ús dels resultats d'aquest projecte, aquesta tasca consisteix en escriure documentació per explicar com instal·lar el sistema, quines dades necessita i com proporcionar-les, com pre-processar i processar aquestes i com rebre notificacions sobre les seves sortides (prediccions fetes en l'entorn).

9.1.2 Taula de temps

Tasca	Assignació de temps prevista (en hores)
Aprenentatge	104
Anàlisi	24
Disseny	72
Implementació	188
Validació	48
Tasques extres	48
Documentació	40
TOTAL	**524**

Taula 12. Taula amb la previsió de temps per a cada tasca

En la planificació d'hores ja s'han inclòs les dues reunions setmanals per a controlar el projecte, i cada tasca ja té inclosa en la seva assignació de temps la duració de les diferents reunions. Les 16 hores restants queden reservades per a desviacions en la planificació inicial.

9.1.3 Restriccions

Per la metodologia elegida (cascada amb retroalimentació) obtenim dependències clares i en ordre entre anàlisi, disseny, implementació i verificació del sistema. Una altra restricció és que no podem començar a crear codi abans d'haver realitzar la fase inicial d'aprenentatge sobre les diferents tecnologies a utilitzar. Hi ha restriccions més específiques, que per ordre de grups de tasques serien les següents:

- **Aprenentatge:**
  - No es pot estudiar com tractar el temps amb el processament de dades abans d'haver après com processar-les.
  - No es pot aprendre quins algorismes de ML tracten el temps abans d'haver estudiat els mètodes més tradicionals.
- **Implementació:**
  - No es pot processar les dades abans d'obtenir-les.
  - No es pot entrenar els models d'aprenentatge automàtic abans d'haver processat les dades per entrenar-los.

9.1.4 Diagrama de Gantt inicial

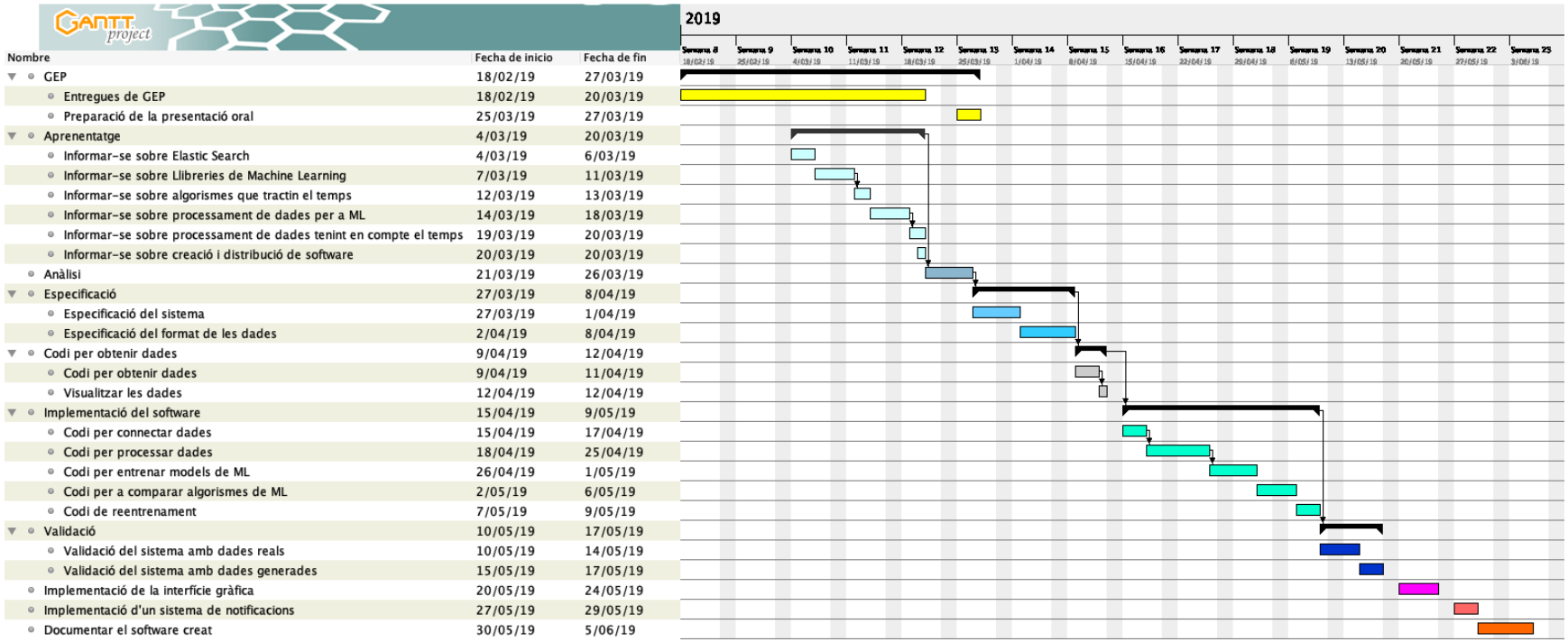

Figura 15. Diagrama de Gantt inicial del projecte.

9.1.5 Recursos destinats al projecte

Els recursos humans necessaris per al projecte són els especificats en la secció **1.4.1** Desenvolupador i **1.4.4** Tutors del Projecte. Només necessitem un ordinador per a desenvolupar aquest projecte.

En quant els recursos de *software,* tot el que s'utilitzarà és gratuït. A part dels programes especificats a la secció **4.1** "Mètodes de seguiment", es farà ús d'altres com el *Sublime Text* per a editar codi o l'intèrpret del llenguatge de programació *Python*.

El cost de cadascun dels recursos s'especifica a la secció **11** Pressupost del projecte d'aquest document.

9.2 Planificació final

9.2.1 Obstacles torbats durant el desenvolupament

El desenvolupament del projecte ha avançat amb grans canvis tant en el Cas A com en el Cas B. Els problemes que han sorgit han sigut de tal envergadura que s'ha canviat l'objectiu de cada cas arribant al plantejament fet en el punt **1.3** d'aquest document. Inicialment, els dos casos pretenien el següent:

- **Cas A:** detectar quan en un servei en el núvol cal modificar-ne la mida del conjunt d'ordinadors que l'integren en funció de la seva càrrega de feina.
- **Cas B:** detectar quan un ordinador dins d'una plataforma al núvol està sobrecarregat de feina i necessita delegar-ne part d'aquesta a un altre.

9.2.1.1 Inviabilitat del Cas A inicial

Si mesurem la *càrrega de feina* que té un núvol en una unitat diferent a "nombre de programes a executar" (com per exemple treball en CPU requerit) és molt difícil saber amb exactitud el valor d'aquesta magnitud. Un programa, en qüestió de demanda de recursos, és a efectes pràctics una caixa negra: no sabem quina és la demanda que tindrà en un instant donat. A més, si té una interfície per a controlar-lo a voluntat, implica que el gast de recursos per a una execució en un moment determinat no té perquè ser igual en el mateix moment però en una altra execució. A més els programes comparteixen màquines on s'executen a la vegada, i aquest fet pot variar-ne substancialment el rendiment de tots ells o d'un subconjunt dels mateixos. Això ens força a canviar l'objectiu del primer cas per a validar *Cronos*.

9.2.1.2 Inviabilitat del Cas B inicial

D'entrada, el Cas B pretén detectar amb aprenentatge automàtic si un ordinador compleix el SLA o no, en termes de qualitat del servei. Per tant, cal obtenir dades per a poder entrenar un model. Això significa aconseguir un conjunt de dades on cada fila d'aquestes correspongui a un valor de "compleix l'SLA" o "no compleix l'SLA".

Per saber si es compleix o no l'esmentat SLA en un servidor donat podríem prendre mètriques com l'ús de la CPU, la memòria, la GPU, el volum de transferència entre disc i memòria, etc. El problema és que aquestes mètriques no són relacionables amb si s'està complint l'SLA o no, ja que un recurs podria estar al 100% del seu ús sense significar que funcioni malament o tingui massa càrrega de feina com per a mantenir estàndard de qualitat esperat. A més un mateix programa al cap de *s* segons de començar la seva execució pot estar fent diferent ús de recursos en diferents llançament, i trobar un valor representatiu amb poc error (si és que existeix) és complex i molt variable (passa igual que en el Cas A).

Un altre problema greu d'aquesta idea inicial és que el que es volia construir era un sistema per a distribuir els programes de forma que es complís de la millor forma possible el SLA (ajuntat els resultats del Cas A i el B) però el problema de la distribució de programes en diferents servidors és inviable. De fet, com explicaré just a continuació és un problema *co-semi-decidible*[4][19] (doncs o retorna fals o mai acaba).

El problema de l'oracle

Suposem que volem distribuir un conjunt de programes que són jocs entre uns servidors de forma *òptima*, on aquesta característica significa que tots aquests programes en la part de servidor funcionen a 30 o més *fps* (suposem que aquest és l'SLA que han de complir). Si tinguéssim un oracle que ens donés la distribució òptima, és possible que volguéssim comprovar si aquest té raó o ens menteix.

Per tal de fer-ho no podem prendre mètriques de cada programa per separat per les raons estipulades abans. Això sí, podem executar el sistema tal com ens ha dit que ho fem l'oracle. Si la combinació no és la òptima, en algun moment algun joc tindrà menys de 30 *fps*, pel que la comprovació haurà acabat de forma negativa i ens haurà mentit. D'altra banda, si cap joc en un moment *t* determinat té menys de 30 *fps*, significa que fins a aquell moment la combinació és òptima, però cal mirar com es comporta per a *t+1*. Per tant, si el suggeriment de l'oracle és òptim, la comprovació mai acaba.

9.2.2 Anàlisi i elecció d'alternatives

Com s'ha explicat ja en punt anteriors, un cop identificats greus problemes en els casos inicialment plantejats ha sigut necessari canviar-los. A continuació exposo les solucions i les decisions preses.

Pel Cas A, d'intentar saber la càrrega de treball d'un núvol en un moment determinat (quantitat difícil de mesurar) s'ha passat a voler predir el nombre d'usuaris, de tal forma que igualment podem intentar saber quin és el nombre de servidors que necessitarem per a

[4] **Problemes co-semi-decidibles:** família de problemes on els algorismes que els intenten solucionar o no acaben mai o retornen fals quan el cas és negatiu. Els problemes ideals per a solucionar són els decidibles, que sempre acaben i retornen el que correspon (veritat per als casos positius, fals per als negatius). Això fa que un algorisme sigui correcte.

mantenir un *cloud* que proporcioni servei a tots els clients (si més no assegurant-ne l'accessibilitat al mateix).

Per altra banda, en el Cas B d'intentar saber si donat un conjunt de programes executant-se en un servidor concret aquests complirien el SLA a la part de servidor s'ha passat a voler predir quants, dels programes executant-se en el núvol en un instant donat, serien d'un tipus determinat i quants d'un altre. D'aquesta forma (i tal com s'ha demostrat en el capítol **9**) els dos casos de validació es complementen i cobren més sentit junts.

Hi ha diverses aplicacions possibles al fet de saber exactament quants i quins programes es voldran executar en un núvol en un moment determinat. En el context del SLA, un exemple clar seria el de distribuir execucions de programes de forma intel·ligent (si s'aconsegueix sortejar els problemes anteriorment d'alguna forma).

9.2.3 Canvis en la planificació

Encara que s'han modificat els casos de validació pels problemes explicats anteriorment, les tasques a realitzar eren suficientment genèriques per a no haver-les de modificar. El que sí s'ha modificat és l'objectiu de cada tasca, perquè cada cas de validació en tenia un de diferent. Per altra banda, finalment no s'ha implementat ni una interfície gràfica per a l'entorn desenvolupat ni un sistema de notificacions, perquè s'ha considerat que era més important invertir el temps de desenvolupament en altres tasques (i igualment, tal com es veu en la taula 13, s'han invertit més hores de les previstes en la implementació).

Dir també que com inicialment s'anaven a consumir dades d'ús de Ludium Lab per a la validació del sistema, m'he informat sobre *Elastic Search* i com consumir dades d'aquest servei, doncs una part de les dades del servei de joc en el núvol estàn emmagatzemades aquí. Finalment ha sigut una tasca realitzada sense repercussió per aquest treball.

9.2.4 Taula de temps final

Tasca	**Hores de treball**
Aprenentatge	120
Anàlisi	20
Disseny	70
Implementació	210
Validació	50
Documentació	50
TOTAL	**520**

Taula 13. Taula amb el temps gastat per a cada tasca

9.2.5 Diagrama de Gantt final

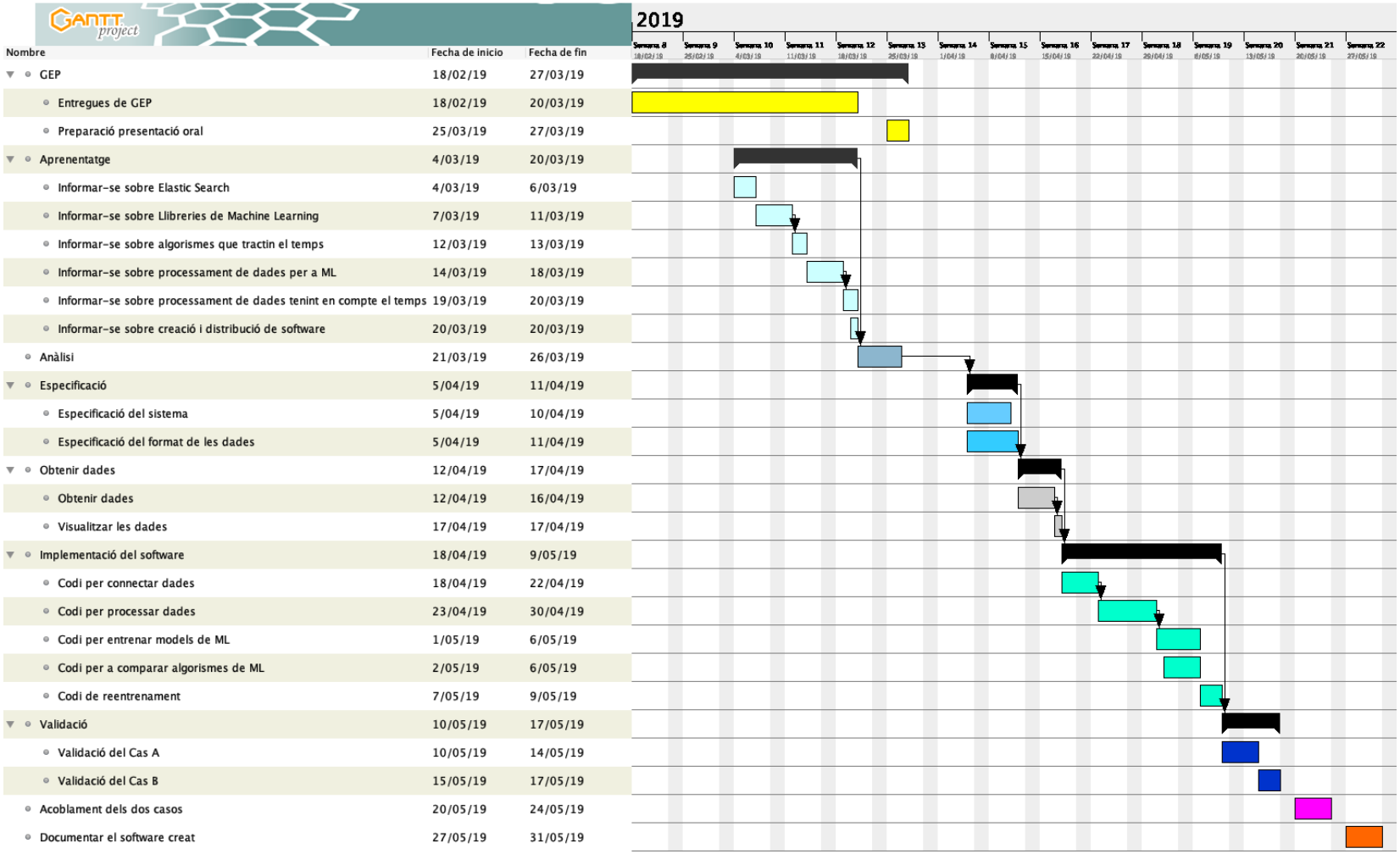

Figura 16. Diagrama de Gantt final del projecte.

10. Pressupost del projecte

10.1 Pressupost inicial del projecte

10.1.1 Identificació de costos

Els costos que s'enumeren a continuació es basen en el temps estimat per cada tasca en la taula 12 i les eines de *hardware* que es faràn servir (apartat **10.1.5**). Tot el software que s'utilitza en el projecte és gratuït.

10.1.1.1 Costos directes

Costos directes de recursos humans

El desenvolupador d'aquest projecte assumeix totes les tasques relacionades amb aquest. Igualment, s'ha desglossat el cost en recursos humans del projecte segons el tipus de tasca a dur a terme i els preus de mercat actuals. El cost per hora del científic de les dades[20] s'ha obtingut de pàgines web en anglès al ser una professió menys nova en els països anglosaxons. Aquest cost ha estat convertit de dòlars americans a euros a raó de 1$ = 0,89€.

Fase	Hores	Rol associat	Preu per hora	Preu total
Anàlisi	24	Cap de projecte	50 €	1.200 €
Disseny	72	Enginyer de software	35 €[21]	2.520 €
Implementació (incloses les tasques extres)	236	Científic de les dades	44,5 €	10.502 €
Validació	48	Cap de projecte	44,5 €	2.400 €
Documentació	40	Enginyer de software	35 €	1.400 €
TOTAL				**18.022 €**

Taula 14. Pressupost de recursos humans

Costos directes de recursos *hardware*

L'ordinador on es desenvoluparà tot el projecte és un Lenovo Y700 [22]. A part d'aquest equip, farà falta un ordinador on desplegar el sistema i un servei en el núvol per a poder validar el programari creat. L'ordinador on executar el programari creat per a la validació pot ser el Lenovo Y700 mateix també[5].

[5] En altres casos, l'ordinador on s'executa el software podria ser o un servidor o un servei.

Equip	Unitats	Preu unitat	Vida útil	Amortització
Lenovo Y700	1	1.010,81 €	5 anys	35,97 €
TOTAL				**35,97 €**

Taula 15. Pressupost de l'equip on desenvolupar i validar el projecte

Costos directes de recursos hardware en el núvol

Inicialment, es volia validar el sistema amb diferents servidors, creant-ne un núvol i veient si l'entorn programat en aquest treball complia els seus objectius. Per a poder verificar els dos casos de validació inicials era necessari tenir tant un sol ordinador com un núvol de servidors (encara que petit, per a veure'n el comportament). Aquest núvol s'havia proposat de compondre'l amb màquines G2 d'Amazon Web Services. En cap altre moment del projecte hi havia previsió de fer ús d'aquests ordinadors. La següent taula mostra el cost de llogar aquests servidors:

Instància	Número d'instàncies	Preu per hora	Hores	Preu total
g2.2xlarge	1	0,62478 [23]	48	29,98 €
g2.2xlarge	5	0,62478 [23]	48	149,94 €
TOTAL				**179,92 €**

Taula 16. Pressupost per al núvol de validació

Fer notar que en la taula superior s'explicita una diferència entre el lloguer del mateix producte dos cops. Això és perquè amb una instància validaríem un cas i amb el núvol de cinc l'altre.

Total de costos directes

En la següent taula s'ajunten els costos directes del projecte i es mostra el total.

Concepte	Cost
Costos directes de recursos humans	18.022 €
Costos directes de recursos hardware	35,97 €
Costos directes de recursos hardware en el núvol	179,92 €
TOTAL	**18.237,88 €**

Taula 17. Total del pressupost directe del projecte

10.1.1.2 Costos indirectes

Els costos indirectes de l'activitat de desenvolupar aquest projecte són els següents:

- En el cas del cost de l'electricitat prenem com a preu del kWh el del dia 7 de març de 2019[24], que és **0.11467€**. Junt amb la potència gastada cada hora i el total d'aquestes, podem calcular el cost de l'electricitat en euros. Si tenim en compte que la potència gastada per l'ordinador és 60W i la de la llum per il·luminar la estància és 116W, sabem que el total de potència gastada per hora és de 0'176 kW. Com que el projecte dura 520h, obtenim el següent cost en euros:

$$CostElectricitat = 520hores \bullet 0{,}176kWh * \frac{0{,}11467euros}{1kWh} = 10$$

- El cost de l'internet ve donat pel cost mensual i el nombre de mesos treballats. Segons es va indicar en l'apartat 5 "Descripció de tasques", es treballarà com a molt 4 mesos (març, abril, maig i juny). Una connexió adient al projecte seria de 100mb per fibra. Per tant, i si el preu mensual de la connexió[25][6] és de 29,95€, en total es gastarà 119,8€.

Per tant, el total dels costos indirectes és **129,8 €**.

10.1.1.3 Fons de contingència

Donat a que el desenvolupador del projecte careix d'experiència en moltes de les tasques a desenvolupar, i encara que el temps reservat per a cadascuna sembla adequat, pot ser una bona pràctica sumar un extra 20% del cost del projecte en contingència. Així, en total afegim **3.673,54 €**.

10.1.1.4 Control de gestió

Els problemes que podria trobar-se el projecte són, principalment, dos:

- Problemes amb l'equip portàtil per a desenvolupar i validar el sistema. Aquest es solucionaria ràpidament reparant l'averia si es pot, o, en el pitjor cas, comprant-ne un de nou.
- Retràs en el desenvolupament del programari en alguna fase determinada. La solució per a aquest contratemps seria afegir més hores de desenvolupament.

Problema	Probabilitat	Cost	Cost total
Espatllament del portàtil	5 %	1.010,81 €	50,5405 €
Retràs en el desenvolupament	10 %	17.398 €	1.739,8 €
TOTAL			**1.790,3405 €**

Taula 18. Cost de les possibles desviacions del projecte

[6] La tarifa elegida té permanència de 3 mesos. Això fa que no haguem de pagar un any sencer de connexió a internet, encara que haguem finalitzat el projecte.

10.1.1.5 Cost inicial total del projecte

En conjunt, en la següent taula es representa el cost total del projecte:

Concepte	Cost
Costos directes	18.237,88 €
Costos indirectes	129,8 €
Fons de contingència	3.673,54 €
Cost del control de gestió	1.852,54 €
TOTAL	**23.893,76 €**

Taula 19. Cost inicial total del projecte

10.2 Pressupost final del projecte

10.2.1 Canvis en els costos. Recàlculs.

Pels canvis donats durant el desenvolupament del projecte, cal recalcular els costos directes. Per començar, els temps gastats en cada tasca ha variat en relació a la planificació inicial de la taula 12 i s'ha de recalcular segons els estipulats en la taula 13. Per altra banda, durant el projecte, i tal com s'ha explicat en l'apartat **10.2** de planificació final, hi ha hagut canvis, principalment en els casos de validació. Canviar els objectius d'aquests ha suposat que per a validar-los no ha sigut necessari fer ús de servidors d'AWS, tal com inicialment sí s'havia proposat en la secció **12.1.1.1**. Això ha creat un estalvi de diners. El recàlcul dels costos directes de recursos humans s'adjunta en la següent taula:

Fase	Hores	Rol associat	Preu per hora	Preu total
Anàlisi	20	Cap de projecte	50 €	1.000 €
Disseny	70	Enginyer de software	35 €[11]	2.450 €
Implementació	210	Científic de les dades	44,5 €	9345 €
Validació	50	Científic de les dades	44,5 €	2.225 €
Documentació	50	Enginyer de software	35 €	1.750 €
TOTAL				**16.770 €**

Taula 20. Pressupost de recursos humans final

Com que els costos dels recursos humans han canviat, també canvien els costos directes totals:

Concepte	Cost
Costos directes de recursos humans	16.770 €
Costos directes de recursos hardware	35,97 €
TOTAL	**16.805,97 €**

Taula 21. Total del pressupost directe del projecte

Un cop modificats els costos directes, també hem de recalcular el fons de contingència del projecte, que és el 20% de la suma dels costos directes i indirectes. Aquesta operació resulta en **3.387,154 €**.

10.2.2 Cost final del projecte

El cost final del projecte un cop recalculats tots els costos és:

Concepte	Cost
Costos directes	16.805,97 €
Costos indirectes	129,8 €
Fons de contingència	3.387,154 €
Cost del control de gestió	1.852,54 €
TOTAL	**22.175,464 €**

Taula 22. Cost final total del projecte

11. Sostenibilitat del projecte

11.1 Matriu de sostenibilitat

La taula mostrada al final d'aquesta secció resumeix en una puntuació del zero al deu que obté el projecte en cadascuna de les dimensions de la sostenibilitat. Donat a que el sistema és capaç de reduir les despeses de les organitzacions usuàries, això significa que la inversió que es fa en ell és justificable, i per tant obté un 9/10 en la dimensió econòmica. Cal dir però que un mal ús del sistema pot augmentar-ne la despesa. Ambientalment, el projecte consumeix molts pocs recursos però un mal ús d'ell, igual que en l'apartat econòmic, podria generar una despesa extra energètica si s'utilitzen unes prediccions dolentes, pel que la valoració final també és 9/10. Per últim, socialment el projecte posa en mans dels seus usuaris les capacitats de l'aprenentatge automàtic de forma més directa, però no n'assegura de cap forma el seu ús responsable i ètic. Aquest és un problema intrínsec a moltes tecnologies, que són una arma de doble tall. Per tant, la valoració en aquest apartat és 8/10.

Dimensió econòmica	**Dimensió ambiental**	**Dimensió social**
9/10	9/10	8/10
TOTAL:	**26/30**	

Taula 23. Matriu de sostenibilitat del projecte

11.2 Dimensió econòmica: reflexió

Econòmicament s'ha de dir que el cost de desenvolupar el projecte, encara que alt, és competitiu. A nivell de recursos materials, requereix d'un ordinador suficientment capacitat per a treballar amb models d'aprenentatge automàtic, pel que és mínim en el nombre d'ordinadors necessitats. En recursos humans el cost és més alt, ja que precisa d'un programador que tingui coneixements de *Machine Learning*, que entengui que és un SLA per entendre les decisions que ha de prendre en relació a la creació del sistema i sàpiga planificar i especificar correctament aquest. Com s'ha especificat abans en la secció **11.2.2**, el cost total del projecte és 22.175,464 €. El cost final és força coherent amb la previsió de despesa inicial del projecte, amb una diferència de 1.718,296 € d'estalvi finalment.

Durant la seva vida útil i depenent de les necessitats de l'organització usuària, el projecte haurà de ser desplegat en diferents ordinadors i potser haurà de ser ampliat i millorat. A nivell de recursos materials, per tant, el cost és mínim. En recursos humans cal per una part mantenir i millorar el sistema, però també cal tenir actualitzats els models d'aprenentatge automàtic dins del sistema. S'ha de vigilar que els models es mantinguin correctes a través del temps, i, si fa falta, re-entrenar-los amb noves dades. Si els models no són suficientment vàlids, la organització usuari pot concórrer en males decisions i pèrdues econòmiques.

Per acabar, cal destacar que la millora en un servei *cloud* que el programari vol aconseguir pot significar que l'organització usuària obtingui més ingressos i en redueixi despeses. Aquest fet fa que en un plaç concret es pugui recuperar la inversió feta en el desenvolupament del projecte i el seu manteniment periòdic, pel que es justifica la viabilitat del projecte.

11.3 Dimensió ambiental: reflexió

L'impacte ambiental de l'electricitat gastada en el projecte durant el seu desenvolupament és baix, ja que tant la construcció com la posada en marxa del projecte requereixen poca energia. Les principals despeses elèctriques són l'ordinador de desenvolupament i llum per il·luminar-se mentre es treballa. En un projecte d'informàtica, i donat a que aquests costos ja són mínims, el que sí que es podria fer és buscar un ordinador amb un consum més eficient per a gastar menys electricitat.

Durant la vida útil del projecte, la despesa més gran en recursos vindrà del desplegament de l'entorn *Cronos* en diferents servidors per a que sigui usat. Cal pensar que el cost més alt es produeix a l'entrenar models, ja que després, gràcies a l'ús de poder automatitzar les prediccions, no cal tenir *Cronos* executant-se sempre en un ordinador. Per tant, un cop creats els models, el cost energètic de tenir l'entorn és petit. El que sí pot propiciar és que la informació obtinguda pel mateix faci augmentar l'electricitat gastada per un servei en el núvol, ja que aquest pot decidir fer ús de més màquines (és a dir, més potència de càlcul) per a poder completar feina en un temps raonable segons el seu SLA. Per altra banda aquest sistema assenta les bases per a una millor gestió dels recursos del servei en el núvol, el que es tradueix en més eficiència. Per aquesta raó globalment es redueix el número de recursos per a mantenir el servei i en redueix la petjada ecològica.

Sí que pot produir-se un augment de la petjada ecològica que produeix el sistema si els models entrenats d'aprenentatge automàtic no funcionen correctament. És el cas de que les prediccions siguin molt més altes que els valors reals, doncs això pot traduir-se en un augment del nombre de servidors en un núvol de forma totalment innecessària. Per això és important mantenir els models actualitzats i suficientment representatius de les dades que es volen predir.

11.4 Dimensió social: reflexió

Personalment aquest projecte m'ha aportat una formació transversal molt interessant en camps tecnològicament punters, com poden ser l'aprenentatge automàtic o els serveis en el núvol. He hagut de reflexionar sobre quines són les necessitats dels *cloud services* i com ajuntar-les amb les capacitats que ens permet tenir l'aprenentatge automàtic. Un cop desenvolupat el sistema, m'he adonat de que aquest sobretot redueix la fricció entre el desenvolupament i l'ús d'un sistema de *machine learning*, i que aquest fet significa que *Cronos* democratitza una mica més aquest conjunt de tecnologies d'intel·ligència artificial, fent-ho més accessible.

El principal beneficiari d'aquest producte seria un servei en el núvol que el volgués utilitzar. Des del punt de vista de l'usuari del servei que faci ús de *Cronos*, aquesta solució no afegeix capacitats noves al servei en el *cloud* però fa que sigui molt més robust. Per tant, és transparent a l'usuari final d'un producte, perquè la intenció és que l'usuari pugui dir que el servei funciona correctament. Donat a la tendència a moure serveis cap al núvol, que aquests siguin estables és una condició interessant i indispensable per a assegurar-ne una bona transició. A més, sempre que tinguin les dades correctes els serveis tindran una manera d'assegurar que el seu SLA es compleix pel que el projecte és necessari i justificat.

Una altre perfil que es podria beneficiar d'aquest entorn desenvolupat és qualsevol persona interessada en l'aprenentatge automàtic. Donat a que amb *Cronos* és molt fàcil modelar de forma ràpida un sistema de *machine learning*, aquest sistema, encara que pensat des de les necessitats del núvol, és totalment usable també en local. Aquest usuari només necessita saber Python per a poder escriure les funcions de pre-processat i processat de dades i seguir les directrius explicades en l'annex de com funciona l'entorn. Per tant, també és un sistema enfocat a un públic amb coneixements de programació però més generalista, no només enfocat al núvol.

Per últim, cal destacar que de la forma en que ha estat dissenyat i construït el sistema és molt fàcil exportar models ja entrenats i crear *back-ups* d'entrenaments costosos, pel que l'usuari es pot protegir de falles en els computadors on executi *Cronos* senzillament cuidant-ne les dades generades per aquest.

Com a reflexió final, dir que l'aprenentatge automàtic, com totes les tecnologies, és una arma de doble tall. Es pot fer servir per a molts propòsits, i recau sobre la moral de l'usuari que aquest sigui un fi noble i ètic. Aquest projecte en concret no anima a ningú a prendre una direcció moral concreta, només ho posa a la seva disposició. Per tant, *Cronos* no canvia l'estat actual moral relacionat amb el *machine learning*, sinó que el manté en les mans de l'usuari que l'utilitzi.

12. Identificació de lleis i regulacions

Tot projecte ha de complir de forma adient les lleis de proteccions de dades d'Espanya[26] (LOPD) i d'Europa[27] (GDPR). El treball exposat en aquest document no recol·lecta dades d'usuaris ni les guarda, pel que està exempt de vigilar aquests punts, encara que les pugui arribar a utilitzar.

El que sí que cal remarcar és que igualment es fa ús de dades i aquestes, encara que fora de l'àmbit del TFG, estan evidentment subjectes a les lleis vigents. Per tant, i com a pas previ a l'alimentació dels diferents algorismes presentats i programats en aquest treball, cal cuidar que la informació utilitzada no infringeixi cap límit legal.

13. Conclusions

13.1 Observacions

En aquest treball s'ha desenvolupat un sistema integral d'aprenentatge automàtic i s'ha demostrat que aquest és funcional, fàcilment manipulable i extensible. S'ha pogut utilitzar el mateix per a crear dos models que ens ajuden a cuidar el compliment del SLA, pel que l'objectiu del treball s'ha complert (encara que hi ha hagut alguns entrebancs sobretot pel que fa al plantejament inicial dels casos de validació i a la intractabilitat d'aquests). L'usuari final d'aquest entorn estarà capacitat per a crear, entrenar i usar ràpidament diferents models d'algorismes d'aprenentatge automàtic d'una forma autònoma, privada i adaptada a les seves necessitats.

A nivell personal, aquest projecte m'ha servit per a complementar per una banda els meus coneixements sobre l'ús de l'aprenentatge automàtic, perfeccionant-ne el coneixement en tres dels algorismes més populars (xarxes neuronals, *random forests* i màquines de vectors suport) i aprenent a modelar dades extraient-ne informació per a fer-les el més òptimes possibles per a alimentar aquests models. Per altra banda, m'ha obligat a aprendre a organitzar un projecte d'una envergadura relativament gran i a dissenyar-lo i implementar-lo de forma que sigui altament modificable, personalitzable i extensible.

13.2 Treball futur

El sistema creat, encara que funcional i útil, té moltes possibilitats d'expandir-se i millorar en el futur. Potser la part més fàcil d'ampliar és la relativa als algorismes d'aprenentatge automàtic: en aquest projecte, i donades les necessitats dels casos que volíem validar, només n'hem implementat tres. Però això no vol dir que no es puguin crear noves classes per a classificar dades o per a realitzar *clustering*. La classe Model de ML proporciona una interfície bàsica que si es segueix permet l'adició de nous mètodes de forma senzilla.

Una altra opció interessant per a *Cronos* seria la construcció d'una interfície gràfica que n'augmentés encara més la reducció de fricció entre els algorismes d'aprenentatge automàtic i qui els vulgui usar. A més, a arrel de la construcció de la interfície, probablement sorgirien noves accions i necessitats que un espera poder realitzar en una finestra, pel que molt probablement s'hauria d'augmentar el nombre de funcionalitats en el que fa a la gestió dels diferents sistemes. Una UI, en definitiva, permetria una gestió més intuïtiva i senzilla de tot el sistema, fent-lo més entenedor.

En la mateixa línia d'augmentar-ne l'accessibilitat, una de les parts més complicades de fer ús de l'entorn actualment és la necessitat d'escriure funcions de pre-processament i processament de dades. La persona que vulgui usar un model no només ha de tenir dades i saber que significa cada paràmetre de l'algorisme d'aprenentatge automàtic que hagi elegit, sinó que ha de ser capaç d'expressar en codi l'extracció de la informació que necessita per a

que el model que construeixi realitzi la tasca que ell vol. Per a fer més fàcil aquesta part, es podria explorar la possibilitat d'afegir eines a l'entorn que facilitin el *feature extraction.*

Per últim, seria també molt interessant poder tenir algun sistema per a re-entrenar de forma automàtica els diferents models que s'estiguin usant per a fer prediccions, amb la fi d'aconseguir models d'aprenentatge automàtic que amb el pas del temps siguin capaços d'entendre les variabilitats dels paràmetres d'un servei (com per exemple, el nombre d'usuaris) i siguin capaços d'inferir quan estan presentant més error del normal per a re-entrenar-se. Aquesta optimització reduiria el manteniment requerit actualment per l'entorn.

Bibliografia

[1] Definició del SLA https://es.wikipedia.org/wiki/Acuerdo_de_nivel_de_servicio

[2] Definició del QoS https://es.wikipedia.org/wiki/Calidad_de_servicio

[3] Explicació de que és l'aprenentatge automàtic https://www.expertsystem.com/machine-learning-definition/

[4] Mohamed, S., Yousif, A., & Bakri, M. (2016). SLA Violation Detection Mechanism for Cloud Computing. International Journal of Computer Applications, 133(6), 8–11. https://doi.org/10.5120/ijca2016907483

[5] Wong TS., Chan GY., Chua FF. (2018) A Machine Learning Model for Detection and Prediction of Cloud Quality of Service Violation. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2018. ICCSA 2018. Lecture Notes in Computer Science, vol 10960. Springer, Cham.

[6] Ibidunmoye, O., Metsch, T., & Elmroth, E. (2016). Real-time detection of performance anomalies for cloud services. 2016 IEEE/ACM 24th International Symposium on Quality of Service, IWQoS 2016, (August), 3–5. https://doi.org/10.1109/IWQoS.2016.7590412

[7] Adamu, H., Mohammed, B., Maina, A. B., Cullen, A., Ugail, H., & Awan, I. (2017). An approach to failure prediction in a cloud based environment. Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017, 2017–Janua, 191–197. https://doi.org/10.1109/FiCloud.2017.56

[8] Aplicació per a monitoritzar indicadors del SLA https://www.manageengine.com/products/applications_manager/sla-management.html

[9] Antonio Arellano Moral (2018). Creación de un sistema de monitorización y seguimiento de errores de aplicaciones ubicuas. http://hdl.handle.net/2117/122591

[10] Creat per l'autor del treball mateix. Inspirat en https://airbrake.io/blog/sdlc/waterfall-model

[11] Llenguatge de programació Python https://www.python.org

[12] Instal·lador de paquets pip https://pypi.org/project/pip/

[13] Documentació de pickle https://docs.python.org/3/library/pickle.html

[14] Documentació d'sklearn http://scikit-learn.github.io/stable

[15] Càrrega dinàmica de moduls https://python-reference.readthedocs.io/en/latest/docs/functions/__import__.html

[16] Explicació de Cron i la crontab https://help.ubuntu.com/community/CronHowto

[17] Explicació sobre el descens de gradinet estocastig https://adventuresinmachinelearning.com/stochastic-gradient-descent/

[18] Portal d'Elastic Search https://aws.amazon.com/es/elasticsearch-service/

[19] Rafel Farré, Robert Nieuwenhuis, Pilar Nivela, Albert Oliveras, Enric Rodríguez, Josefina Sierra (2009). Notas de Clase para IL. Deducción en Lógica de Primer Orden.

[20] Sou dels científics de les dades https://www.upwork.com/hiring/data/how-much-hire-data-scientist/

[21] Sou de l'enginyer de software https://nosotros.infojobs.net/prensa/notas-prensa/ranking-de-los-puestos-mejor-pagados-en-espana

[22] Ordinador portàtil Lenovo Y700 https://www.amazon.es/Ideapad-Y700-15ISK-Portátil-i7-6700HQ-GeForce/dp/B01D1WVNFE?psc=1&SubscriptionId=AKIAJWI4VHIVLDLAD5TQ&tag=uvl483598-21&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B01D1WVNFE&keywords=lenovo%20y700

[23] Cost dels servidors d'Amazon Web Services https://aws.amazon.com/es/emr/pricing/

[24] Cost mig del kWh del 7 de març de 2019 https://tarifaluzhora.es/?tarifa=normal&fecha=2019-03-07

[25] Cost de fibra 100mb amb la companyia Lowi https://comunicacion.kelisto.es/landings/lowi/lowi-fibra-100-mb?section=dynamic

[26] Ley Orgánica de Protección de Datos (LOPD) https://www.boe.es/eli/es/lo/1999/12/13/15/con

[27] General Data Protection Regulation (GDPR) https://gdpr-info.eu

Annexos

A. Instal·lació de *Cronos*

B. Ús de *Cronos*

   1. Creació de sistemes. Exemples de funcions de pre-processat i processat
   2. API per a la integració amb altres programes

A. Instal·lació de *Cronos*

Instal·lació de Python 3

El programa per a instal·lar Python 3 i les instruccions per a cada sistema operatiu poden ser trobades en https://www.python.org/downloads/. Cronos necessita una versió de Python igual o superior a la 3 per a funcionar correctament.

(Opcional) Creació d'un l'entorn virtual de Python

La creació d'un entorn virtual per a Python ens permet aïllar les llibreries instal·lades en aquest entorn sense fer canvis en l'entorn per defecte d'aquest llenguatge en el sistema.
La comanda és `python3 -m venv ruta/On/Crear/venv`. S'ha de modificar posteriorment l'arxiu *configurePredict.sh* de l'entorn per a que apunti a aquest entorn virutal.

Instal·lació de les llibreries per a executar correctament l'entorn

Es fa amb la eina `pip`. Cal cridar a dues comandes: `pip install matplotlib` i `pip install sklearn`. La primera llibreria és per a poder crear gràfics, i la segona per a poder fer ús de diferents algorismes d'aprenentatge automàtic. Si s'ha decidit fer ús d'un entorn virutal, cal entrar-hi per instalar-hi les llibreries en ell.

Instal·lació de *Cronos*

Per a poder executar *Cronos* i un cop fets tots els passos anteriors, només cal copiar el sistema a la ruta desitjada. Pot fer falta modificar el configurePredict.sh amb la ruta per a poder executar les prediccions automàtiques.

Després, per a executar Cronos cal cridar a la comanda `python systemsMain.py`.

B. Ús de *Cronos*

1. Creació de sistemes. Exemples de funcions de pre-processat i processat

Creació de sistemes

Per a crear un nou sistema d'aprenentatge automàtic en *Cronos* en general només cal seguir les instruccions que es proporcionen vía consola. Tot hi així hi ha alguns punts que s'han d'aclarir. Abans de poder entrenar un sistema, cal:

- Haver elegit quins són els fitxer d'entrenament, validació i entrada.
- Haver especificat quin és el fitxer de sortida.
- Haver especificat les funcions de pre-processament, processament i el fitxer que les conté.

Aquests paràmetres s'elegeixen en les funcions 4, 5 i 6 de qualsevol sistema.

Exemple de funcions de pre-processat i procesat

```
def preprocess(data):
      print('preprocessing...')
      prePorcessedData = []
      for d in data:
            aux = d
            for i in range(0, len(d)):
                  if d[i] == '':
                        aux[i] = 0.0
            prePorcessedData += [aux]
      return data

def process(data):
      print('processing...')
      X = []
      Y = []
      for d in data:
            aux = []
            for i in range(0,len(d)):
                  if i == (len(d) - 1):
                        Y += [float(d[i])]
                  else:
                        aux += [float(d[i])]
            X += [aux]
      return X, Y
```

2. API per a la integració amb altres programes.

Important la funció *predictJob()* de la classe *systemsMain.py* es pot usar aquesta per a que els sistemes entrenats i habilitats per a prediccions automàtiques llegeixin les seves dades d'entrada, realitzin prediccions i escriguin els resultats en els seus fitxers de sortida. Es pot cridar a un conjunt concret de sistemes per a que realitzin el procés, passant com a argument de *predictJob(llista)* una llista amb els noms dels sistemes. Si la llista és buida, es crida a tots els sistemes (és com si no passessim cap paràmetre).

Per tant, escrivint en els fitxers d'entrada, cridant a la funció *predictJob(llista)* amb el conjunt de sistemes que ens interessa i recollint els resultats dels fitxers de sortida podem fer ús de Cronos en altres aplicacions.